СОДЕРЖАНИЕ

Часть I. УСЛОВИЯ ЗАДАЧ

Часть II. ПЕРВАЯ ПОМОЩЬ

Часть III. И ЕСЛИ ВЫ ВСЕ ЕЩЕ НЕ НАШЛИ РЕШЕНИЯ

ББК 22.18
А85
УДК 519.68

JEUX

et casse — tête

A

PROGRAMMER

JACQUES ARSAC

DUNOD

Арсак Ж. Программирование игр и головоломок: Пер. с франц.— М.: Наука. Гл. ред. физ.-мат. лит., 1990.— 224 с.— ISBN 5-02-013959-9.

Рассматриваются способы программирования различных занимательных игр и головоломок с числами, геометрическими фигурами и др. Изложение большинства игр и головоломок ведется в несколько этапов. Сначала разъясняется сама постановка задачи и требования, предъявляемые к алгоритму ее решения.

В следующем разделе книги обсуждается сам алгоритм и возможные пути его реализации.

В конце книги по многим играм и головоломкам даются наброски их программной реализации. Используемый при этом язык типа Паскаля допускает перевод на другие широко распространенные языки программирования.

Для начинающих программистов, студентов вузов и техникумов.

Ил. 41. Библиогр. 15 назв.

А $\frac{1404000000-036}{053(02)-90}$ 129-89

ISBN 5-02-013959-9

ПРЕДИСЛОВИЕ

«Играйте с компьютером, это захватывает...»

«Нет ничего увлекательнее создания собственных компьютерных игр»...

Это — два привлекательных лозунга, взятые из рекламы микрокомпьютеров. Поэтому самое существенное — это создавать игру.

КТО САМ ПРОГРАММИРУЕТ СВОИ КОМПЬЮТЕРНЫЕ ИГРЫ, НАСЛАЖДАЕТСЯ ДВАЖДЫ *)

Я уже проводил небольшое расследование в SICOB в 1984 году. Во многих торговых предприятиях я представлялся как отец, который хочет подарить своему сыну микрокомпьютер. Мне объяснили, что именно нужно купить и во сколько это мне обойдется. Затем я спрашивал об играх. И вот что, приблизительно, там происходило.

« —Так он сможет со всем этим играть?

— Да нет же, месье. Чтобы он мог играть с компьютером, он должен покупать кассеты с играми.

— Кассеты с играми? Я думал, что компьютер... А сколько стоит кассета?»

Продавец называет мне цену, которую вы, должно быть, знаете. Я отвечаю:

«— И сколько игр на кассете?

— Одна, месье.

— Но тогда двадцать игр стоят дороже самой машины! А я слышал от сведущих людей, что на компьютере интереснее всего создавать игры самому.

— Для этого, месье, нужно уметь программировать.

— Но по телевизору твердят, что это очень просто. Мой сын, должно быть, может научиться?

---

*) Я здесь совершаю плагиат по отношению к поговорке жителей плоскогорья Высоких Вивар, которая звучит так: кто сам пилит свои дрова, согревается дважды.

— Да, месье. У нас есть кассета с Бейсиком.
— Кассета с Бейсиком! Бейсик — это и есть программирование?
— Нет, месье. Чтобы уметь программировать, нужна еще вот эта кассета с языком ассемблера...»

Я удалялся, исполненный отвращения, жалкий истец отцов семейств! В другом торговом заведении я отважился утверждать, что я слышал от сведущих людей, что некоторые журналы публикуют совершенно готовые игровые программы.

«— Да, это так. Но эти игры плохо сделаны. Они не являются профессиональной работой, как наши кассеты, где вывод изображения на экран тщательно отработан. Кроме того, очень неприятно набирать тексты этих игр на клавиатуре. Так что все это не очень серьезно, что бы об этом ни говорили...»

На этот раз я поверил, что мне говорят правду. Опубликованные таким образом игры весьма посредственны по качеству. Я не поленился внимательно прочесть некоторые из них (см., например, головоломку 28). Чаще всего они плохо составлены и так плохо прокомментированы, как будто автор хочет помешать читателю понять их. Стратегия этих игр иногда более чем примитивна: я видел игру ТИК—ТАК—ТОК, где компьютер случайным образом выбирал свой игровой ход... Это издевательство над читателями!

Во всем этом совершенно отсутствует творчество. Реклама трубит: «Создавать — это увлекательно!». Но на практике все делается так, как будто подростки неспособны что-либо создавать. И им не помогают начать делать что-нибудь творческое.

Чтобы восполнить этот пробел, я и написал эту книгу. Многие подростки обучены программированию — в лицее (причем не только в таких лицеях, которые официально предоставляют обучение со специализацией по информатике, устраивая курсы по программированию), в клубах и на каникулах — с родителями, с друзьями, по книгам... Но вот что программировать на компьютере? Игры? Какие? Примеры, публикуемые в журналах и книгах, непонятны. Как они могут хоть на что-то вдохновить? А если и посмотреть вокруг, что можно найти? Шахматы, Отелло, вошек? Это слишком трудно...

Поэтому я собрал здесь описания некоторых игр, которые есть в продаже, чаще всего на Бейсике, для ко-

торых не нужны графические возможности компьютера. Я сам все эти игры реализовал на микрокомпьютере, имеющем только алфавитно-цифровой черно-белый экран и не имеющем ни светового пера, ни мыши, ни планшета. Трудность программирования обозначается звездочками «*». Начинайте с игр без звездочек. К остальным вы перейдете позже. Каждая игра допускает многочисленные вариации, которые вы легко изобретете.

Для того чтобы менее мужественные читатели могли избежать искушения сразу же читать все подробности изготовления программ, в первой части книги условие игры обычно излагается вместе с некоторыми указаниями. Если этих указаний вам недостаточно, иначе говоря, если после размышления и возможных обсуждений с товарищами вы не видите, как тронуться с места, вас выведет из затруднений «первая помощь». Если же вы опять упретесь в какую-нибудь трудность, то третья часть книги должна позволить вам достичь цели. Смелее вперед: СОЗДАВАЙТЕ.

Игры привлекательны не только для молодых людей от 14 до ? (а до скольких?). Я знаю многих взрослых, любящих играть (и сам люблю). У меня много коллег, для которых программирование и есть игра — как для меня. Я написал эту книгу для всех молодых людей от 14 до 77 лет, которые любят играть с компьютером или которые любили бы это делать, если бы средство для этого им было предоставлено.

Серьезные журналы публикуют математические развлечения для «порядочных людей». Продаются книги с логическими головоломками. Продаются обзоры стратегий. Применение компьютеров полностью обновило весь этот круг вопросов. Всем тем, кто интересуется головоломками, я предлагаю здесь новую форму головоломок: задача, решение которой нужно найти на компьютере,— это настоящая головоломка. Они бывают трех основных типов:

некоторые головоломки интересно программировать (например, зашифрованные операции, господин S и господин P, пентамино...);

некоторые задачи имеют очень простой вид, но доведение решения до программы является настоящей головоломкой (например, переставить две части вектора, найти наибольший белый прямоугольник в решетке кроссворда). Это — совершенно новый тип задач, тесно связанный с компьютером;

наконец, последний тип — настолько трудный, что я уменьшил набор его примеров, выбрав их из поистине неисчислимого множества,— связан с тем, что я говорил в начале. Вам дана программа (в нашей книге — написанная настолько хорошо, насколько это возможно), но без комментариев. Скажите, что она делает.

Трудность головоломки обозначается вопросительными знаками. Есть головоломки с довольно простыми решениями, но деликатным программированием: «?***», и есть ужасные головоломки с легким программированием: «???». Выбирайте!

Эта книга будет полезна и всем тем, кто занимается подготовкой подростков по информатике: преподавателям лицеев и колледжей, руководителям клубов или каникулярных занятий.

Я не привожу никаких решений, за исключением нескольких настолько классических случаев, что их решение исследовано всюду, или задач, решение которых является образцом, который часто встречается и который нужно знать. Изучите сначала их.

Первый раздел — немного особенный. Каковы бы ни были ваши склонности — начните с него; он даст вам инструмент, необходимый для почти всех игр и немалого числа головоломок (формирование ситуаций случайным образом).

Если вы уже перелистали книгу, то вы, может быть, заметили в ней и кое-что, похожее на математику. Признаюсь! Несколько головоломок в ней — арифметические, и их решение требует вычислений. В бо́льшей части эта книга доступна и людям, не имеющим никакой специальной математической подготовки, даже тем, кому вся эта математика внушает страх и отвращение.

Я сказал все. Теперь — вам ИГРАТЬ, вам СОЗДАВАТЬ...

ОБОЗНАЧЕНИЯ

Вот конструкции, используемые в программах этой книги.

Оператор присваивания. В нем используется знак «:=»

$i := i + 1$

Вот его аналоги на других языках:

Бейсик: LET I = I + 1
LSE: I ← I + 1
Паскаль: I := I + 1

Условный оператор имеет вид

ЕСЛИ условие ТО последовательность операторов
КОНЕЦ_ЕСЛИ

При работе условного оператора вначале проверяется условие. Если оно имеет значение ИСТИНА, то выполняется последовательность операторов, заключенная между ТО и КОНЕЦ_ЕСЛИ. КОНЕЦ_ЕСЛИ играет роль закрывающей скобки, избавляющей от применения разделителей DEBUT FIN, как на LSE, или BEGIN END, как в языке Паскаль. При работе оператора

ЕСЛИ условие ТО последовательность операторов
ИНАЧЕ последовательность операторов
КОНЕЦ_ЕСЛИ

вначале проверяется условие. Если оно имеет значение ИСТИНА, то выполняется последовательность операторов, заключенная между ТО и ИНАЧЕ, а если условие имеет значение ЛОЖЬ, то выполняется то, что содержится между ИНАЧЕ и КОНЕЦ_ЕСЛИ. Снова, как и выше, нет нужды в DEBUT FIN.

Цикл

ПОКА условие ВЫПОЛНЯТЬ
последовательность операторов
ВЕРНУТЬСЯ

выполняет последовательность операторов, заключенную между скобками ВЫПОЛНЯТЬ — ВЕРНУТЬСЯ, пока условие справедливо. Он эквивалентен циклу LSE

FAIRE номер строки ПОКА условие
последовательность операторов
n замыкающая строка

или циклу на языке Паскаль

```
WHILE условие DO
    BEGIN последовательность операторов END
```

Цикл

```
ВЫПОЛНЯТЬ
    последовательность операторов, содержащая
    слово КОНЧЕНО
ВЕРНУТЬСЯ
```

работает так:

Последовательность инструкций, заключенная между скобками операторов ВЫПОЛНЯТЬ — ВЕРНУТЬСЯ, повторяется неограниченно. Слово КОНЧЕНО означает, что цель цикла достигнута, повторяемая работа закончена. На этом цикл останавливается и программа продолжается со следующего за циклом оператора В английских книгах и статьях вместо КОНЧЕНО обычно пишут EXIT: выйти из цикла (также сделано и в языке Ада). Но EXIT вызывает идею действия: выхода. Я предпочитаю ему слово КОНЧЕНО, которое лучше отражает идею не действия, а ситуации: я достиг цели цикла, с ним все кончено....

Простых эквивалентов этого цикла на Бейсике, LSE или Паскале нет. Можно применить операторы ALLER EN или GO TO для симуляции такого цикла.

— На Бейсике можно использовать дополнительную переменную Z:

```
FOR Z = 1 TO 0 заменяет ВЫПОЛНЯТЬ
LET  Z = 0      заменяет КОНЧЕНО
NEXT  Z         заменяет ВЕРНУТЬСЯ
```

Кроме того, нужно перепрыгнуть в цикле все, что стоит после слова КОНЧЕНО, т. е. после оператора LET Z = 0. Так как это можно сделать с помощью GO TO, то я считаю предпочтительным использовать таким образом GO TO для циклов. Если ваш язык не структурирован, то красивых циклов вы никогда не получите...

— На языке Паскаль используйте булеву переменную *z*, которой до начала цикла присвоено значение TRUE, и тогда цикл примет вид

```
WHILE z DO BEGIN END
```

Слово КОНЧЕНО придется заменить оператором *z* := FALSE, включенным в конструкцию так, чтобы сделать этот оператор последним выполняемым оператором цикла. Если структура языка нехороша...

Цикл

```
ДЛЯ i := exp 1 ШАГ exp 2 ДО exp 3 ВЫПОЛНЯТЬ...
ВЕРНУТЬСЯ
```

повторяет последовательность операторов, заключенную между ВЫПОЛНЯТЬ и ВЕРНУТЬСЯ, придавая *i* значения из арифметической прогрессии с разностью exp 2 (постоянная величина в данном цикле), начиная с exp 1 и останавливаясь на exp 3. Если шаг равен 1, то фрагмент ШАГ 1 можно опустить.

ЧАСТЬ I
УСЛОВИЯ ЗАДАЧ

1. СЛУЧАЙНЫЕ ЧИСЛА

Генерация случайного числа

Можно сделать из этого настоящую головоломку: написать программу, выполнение которой на компьютере дает число, случайным образом расположенное в данном интервале, например, между 0 и 1. Но это невозможно.

Некоторые языки содержат функцию, значение которой есть непредсказуемое число в данном интервале. Если ваш компьютер использует LSE, достаточно набрать на клавиатуре

```
?ALE(0)
```

чтобы получить в ответ непредсказуемое число между 0 и 1, которое может рассматриваться как полученное случайным образом.

На языке Бейсик команда RND(0) дает тот же эффект при условии, что предварительно выполнена инструкция RANDOM. Но Бейсик — это скорее общее имя для целого класса языков, чем обозначение совершенно определенного стандартизованного языка, не меняющегося от одной машины к другой. Так что сверьтесь с описанием к вашему компьютеру...

Если используемый вами язык допускает описанные выше или аналогичные возможности, то получить случайное число в интервале (0, 1) — это никакая не головоломка, это тривиально.

Но если в языке такой возможности нет, то это больше чем головоломка, это невозможно. Предположим, что мы сделали программу, производящую такое число. Эта программа не может иметь исходных данных, иначе это не она вытаскивает случайное число, а именно вы при введении данных... Если же у нее нет данных, то она действует, исходя из констант. Но тогда нет переменных

элементов, и последовательные запуски программы дают совпадающие результаты. Как же вы получите с помощью такой программы случайное число? *)

Поэтому если ваш компьютер не допускает функции, дающей стохастическое число, я вижу только одно решение **): введите сами случайное число в ваш компьютер. Как это сделать? Вот предложение. Вы берете колоду из 52 карт, перетасовываете. Затем вы делите колоду на верхнюю и нижнюю части и берете из нижней части три верхние карты. Небольшая и очень простая программа читает три целых числа x, y, z. В качестве значений этих целых чисел вы задаете численные значения трех выбранных карт, считая туза за 1, валета — за 11, даму — за 12, короля — за 13. Компьютер вычисляет значение

$$(((x-1)/13+y-1)/13+z-1)/13.$$

Например, если вы достанете, как только что случилось со мной, семерку бубен, десятку червей и еще шестерку бубен, то

$$x=7 \quad y=10 \quad z=6$$

и компьютер получит 0.440601.

Это значение не является воистину непредсказуемым. Как только вы достанете карты, вы уже сможете понять порядок величины результата. С другой стороны, таким способом вы не сможете получить более $13^3 = 2197$ различных чисел. Но этого на самом деле достаточно для приложений, которые мы рассматриваем в этой книге. А если вы и в самом деле хотите получить что-либо непредсказуемое, прочтите следующий раздел.

---

*) Строго говоря, эти рассуждения применимы к любой программе, написанной на любом языке, если только эта программа не использует никакой внешней информации в качестве исходных данных. В качестве такой внешней информации удобнее всего использовать что-нибудь связанное с временем: число изменений напряжения в сети с момента последнего включения вашего компьютера или число секунд с момента его покупки, если ваш компьютер снабжен внутренними энергозависимыми часами (на литиевой батарейке), и т. п. Обычно, на каком бы языке вы ни работали, у вас есть возможность прочесть показания внутренних часов компьютера (посмотрите в документации, как работать с таймером).— *Примеч. ред.*

**) См. предыдущую сноску.— *Примеч. ред.*

Непредсказуемые числовые последовательности

Редко бывает нужно получить только одно случайное число. Чаще нужно получить много таких чисел. Большая часть игр, представленных в этой книге, требует, чтобы играющий с компьютером по ходу игры встречался, сообразно с предложенными правилами, с непредсказуемыми ситуациями. Нужно уметь порождать такие ситуации.

Поэтому нужно иметь возможность построить такую последовательность чисел, чтобы переход от одного числа к другому определялся простыми вычислительными правилами, но чтобы в то же время результат было трудно предсказать.

Может случиться, что используемый вами язык предоставляет эту возможность непосредственно в виде одной из конструкций языка.

Так, на LSE команда ALE (x) дает число, лежащее в интервале (0, 1), значение которого зависит от x, но непредвиденным образом и, кроме того, не специфицированным в языке: значение будет различным на разных машинах. Если вы трижды зададите один и тот же вопрос

?ALE(0.1)

вы каждый раз получите один и тот же ответ, но между ALE(0.1) и ALE(0.2) нет простого соотношения.

На Бейсике функция RND играет ту же самую роль. Она порождает непредсказуемую последовательность, значения которой зависят только от начального числа — оно одно и то же для данного компьютера. Инструкция RANDOM дает случайное число и ставит его начальным элементом последовательности, что позволяет порождать различные последовательности.

Может быть, интересно посмотреть, как можно строить подобные последовательности. Вот метод, предложенный А. Энгелем [ENG]. Если x — число между 0 и 1, то следующий за x элемент последовательности есть

дробная_часть $((x + 3.14159)^8)$

Конечно, восьмая степень вычисляется тремя последовательными возведениями в квадрат! Она дает число между 9488 (для $x = 0$) и 86564. Очень небольшое изменение x вызывает сильное изменение $(x + \pi)^8$, и, в частности, оно может перейти через ближайшее целое, так что новое значение (дробная часть результата) может оказаться меньше предыдущей.

Возьмем, например, $x = 0.52000$; тогда

$$(x + 3.14159)^8 = 32311.5437$$

так что за 0.5200 следует 0.5437.

Но для $x = 0.52005$ имеем

$$(x + 3.14159)^8 = 32315.0736$$

и за 0.52005 следует 0.0736.

Так как мы берем дробную часть, то полученное число, разумеется, лежит между 0 и 1.

Упражнение 1. Поведение последовательности.

Речь идет о том, чтобы увидеть, как ведут себя числовые последовательности, порожденные таким образом. Для этого вычислим большое число членов последовательности, порожденной своим первым элементом. Поместим каждый из этих членов в один из 50 интервалов длины 0.02, составляющих интервал от 0 до 1. Выведем число членов последовательности, попавших в каждый из этих интервалов. Если числа из последовательности равномерно распределены в интервале (0,1), мы должны будем обнаружить, что их количество в разных интервалах имеет ощутимую тенденцию к постоянству.

Составьте программу для проверки этого утверждения. Начальное значение может, например, вводиться в начале каждого вычисления.

Упражнение 2. Поиск других последовательностей.

Число π, использованное при вычислении наших последовательностей, не обладает никаким специальным свойством, и можно спросить себя, действительно ли выбор этого числа является наилучшим возможным. Числа $(x + \pi)^8$ довольно велики, а берем мы от них только дробную часть. При этом мы отбрасываем значащие цифры целой части, и — поскольку вычисления на компьютере проводятся с фиксированным количеством значащих цифр — на дробную часть остается относительно небольшое количество цифр. Предположим, что числа представляются с помощью 24 двоичных цифр. Нужно 14 двоичных цифр, чтобы записать 9488, так как

$$2^{13} = 8192 < 9488 < 2^{14} = 16384$$

и 17 цифр, чтобы записать 86564, так что остается лишь от 7 до 10 двоичных цифр на дробную часть.

Используя $(x + a)^8$ вместо $(x + \pi)^8$ с меньшим значением a, можно ожидать сохранения большего коли-

чества значащих цифр для дробной части. Но нельзя взять a слишком близко к 1, так как тогда распределение чисел в интервале (0, 1) окажется плохим. Можете ли вы объяснить, почему?

Например, почему нельзя взять $a = \sqrt[8]{2}$?

Если вы сделали упражнение 1, вы располагаете программой для проверки случайных чисел. Измените ее так, чтобы она осуществляла чтение

— постоянной a,

— начального значения последовательности.

На своем микрокомпьютере я выяснил, что $a = 1.226$ дает достаточно хорошие результаты. Но это наблюдение может меняться от машины к машине, так как все это очень чувствительно к способу, которым осуществляются умножения: в последней двоичной цифре результата умножения есть неопределенность, существенно влияющая на рассматриваемый процесс.

Азартные игры

Теперь вы должны быть в состоянии получать последовательности случайных чисел. Либо эта возможность есть в используемом вами языке, либо вы можете построить непредсказуемую последовательность чисел методом, описанным в предыдущем разделе.

Упражнение 3. «Орел» или «решка».

Я не осмеливаюсь предложить это как игру; это скорее упражнение, чтобы научиться использовать случайные числа. Составьте следующую программу:

— она спрашивает вас, что вы загадали, «орла» или «решку», и читает ваш ответ;

— она порождает случайное число и затем сообщает вам, выиграли вы или проиграли.

Единственная трудность: генератор случайных чисел дает вам, быть может, число, содержащееся между 0 и 1 (это так в случае функции ALE языка LSE и для генератора из разд. 1.2. Имеются противоречивые сведения о языке Бейсик, возможности которого существенно меняются от машины к машине). Следовательно, нужно перейти от вещественного числа в интервале 0 : 1 к чему-либо принимающему не более двух значений, например 0 и 1. Вы сопоставляете по своему усмотрению «орла» одному из них, а «решку» — другому.

Если генератор случайных чисел действует не лучшим образом, то игра может оказаться нечестной, и одна из возможностей — «орел» или «решка» — может выпадать чаще, чем другая.

Сделайте программу, реализующую большое число испытаний, и подсчитайте число выпаданий орла.

Упражнение 4. Игральные кости.

Вместо игры в «орла» или «решку» заставьте компьютер играть в кости. Напишите программу, симулирующую большое число выбрасываний двух костей, и подсчитайте, сколько раз будет выпадать каждая комбинация от 2 до 12. Знаете ли вы, сколько раз должна выпасть каждая из них, если генератор случайных чисел идеален, а число бросаний действительно велико?

Перед вами — та же задача, что и в предыдущем упражнении: перейти от некоторого числа в интервале (0, 1) к целому числу от 1 до 6 включительно. Но если вы знаете, как сделать это для 2, то вы сможете сделать и для 6...

Игра 1. Фальшивые кости.

Ну да, такое еще бывает; есть еще такие люди, которые мошенничают и используют поддельные кости. Нужно быть в состоянии заметить это и, как в любом хорошем вестерне, устроить грандиозную драку с мошенником.

Здесь мошенником пусть будет компьютер. Он играет одной-единственной костью и бросает ее столько раз, сколько вы требуете. Он дает вам число выпаданий единицы, двойки, ..., шестерки. Вы сообщаете ему, верите ли вы, что кости поддельные, и если да, то какая грань выпадает чаще других. Компьютер отвечает вам, выиграли вы или проиграли, и случайным образом оценивает ваш выигрыш. Совершенно ясно, что если вы потребуете 10 000 бросаний, то у вас будет больше шансов обнаружить истину, чем если вы потребуете 20 бросаний...

Нужно решать две задачи: компьютер должен выбрать — подделывать кости или не подделывать, и если он их подделывает, то он должен решить, какая грань будет встречаться чаще остальных.

Вспомогательная задача: выбрать функцию оценки для выигрыша.

? Игра 2. Стратегия для одной игры в кости.

Ж.-К. Бейиф [BAI] предложил игру в кости с двумя игроками. Каждый игрок в свою очередь хода бросает кость столько раз, сколько хочет. Если он не выбрасывает единицу, то он записывает за этот ход сумму выпав-

ших за бросания этого хода очков. Если же он выбрасывает единицу, то он не записывает ничего (и его ход кончается с выбрасыванием единицы). Выигравшим считается тот, кто первым наберет (или превысит) 100 очков.

Составьте программу, которая позволит человеку играть против компьютера. Эта программа реализует бросание кости. На своем ходе она честна и не мошенничает. На вашем ходе она бросает кость и сообщает, что выпало, а вы требуете следующего бросания, если вы хотите играть дальше.

Задача о стратегии ясна. Вы можете, например, бросать кость ровно один раз. У вас хорошие шансы увеличить свою сумму, но на небольшое число очков (от 2 до 6). Если вы делаете несколько бросаний, вы увеличиваете шанс получить большую сумму, но вы увеличиваете и риск выбросить единицу. Стратегия играющего против компьютера — это его проблема, и программа компьютера во всех этих рассмотрениях не участвует. Она играет по команде человека, он говорит, хочет ли он продолжать — под его личную ответственность.

Напротив, программа должна быть снабжена стратегией для управления игрой компьютера. Возможностей много. Выбирать следует вам.

Программирование этой игры представляет двоякий интерес:

— нужно придумать стратегию для компьютера;

— у вас есть возможность экспериментировать. Если компьютер снабжен некоторой стратегией, то вы можете играть против него с другой стратегией и посмотреть, кто выигрывает...

Вы можете также захотеть переиграть партию с тем же началом, что и в предыдущей партии, но вводя в вашу игру изменения, чтобы изучить последствия. Это приводит к новому понятию.

Воспроизводимая непредсказуемая последовательность

Вы научились порождать последовательности непредсказуемых чисел, или, допуская неточность речи, принятую в информатике, случайных чисел (эти последовательности совершенно не случайны: они полностью детерминированы, но, поскольку мы не можем найти простого способа перехода от данного числа к следующему и поскольку эти числа приблизительно регулярно размещены в промежутке 0 : 1, то они производят впечатление слу-

чайности). Каждое число в этой последовательности зависит только от предыдущего числа. К тому же, как и выше, вы можете получить и в самом деле непредсказуемое число, задавая компьютеру значения трех карт. Вы заставляете его вычислить значение, определенное в разд. 1.1, затем вы берете следующее за этим значением либо с помощью функции ALE или RND вашего компьютера, либо с помощью метода, описанного в разд. 1.2.

Эти последовательности случайных чисел таковы, что каждое число в последовательности зависит только от предшествующего ему и задание начального элемента последовательности полностью определяет последовательность. При отправлении из одной и той же точки два последовательно проведенных вычисления дают одинаковые последовательности. Таким образом, вы не только можете получить в играх непредсказуемые ситуации, но и воспроизвести их столько раз, сколько вам нужно. Для этого нужно, чтобы программа требовала ввести исходное значение последовательности. Я считаю удобным вывести приглашение приблизительно такого рода:

ВВЕДИТЕ ТРЕХЗНАЧНОЕ ЦЕЛОЕ

затем прочесть значение x этого целого и взять в качестве начального значения случайной последовательности число $x/1000$.

Исходя из генератора случайных чисел, можно легко построить последовательность целых чисел. Название функции, порождающей случайные числа, меняется от языка к языку; назовем ее

ale (x)

— это функция, сопоставляющая x, $0 \leqslant x < 1$, следующее за ним число

$0 \leqslant \text{ale}\ (x) < 1.$

Построим теперь последовательность неотрицательных целых чисел, меньших данного числа n. У нас есть две возможности.

1. Мы порождаем последовательность случайных чисел в интервале (0, 1) и для каждого из чисел последовательности получаем соответствующее целое

$x := \text{ale}\ (x)$, $p = \text{целая_часть}(n * x)$.

Различные значения x могут давать одно и то же значение p, так что элемент, следующий за p в последова-

тельности целых, не определяется каким-либо предсказуемым образом. Вообще говоря, данное значение p может иметь несколько последующих значений. Маловероятно, что эта последовательность окажется периодической. Это заведомо случится, если последовательность x, определяющая ее, периодична (а это бывает, хотим мы этого или не хотим).

2. Пусть p дано; тогда p/n лежит между 0 и 1. И элемент, следующий за p, можно определить формулой

$$p := \text{целая_часть}\ (n * \text{ale}\ (p/n)).$$

Здесь элемент, следующий за p, полностью определен числом p, и эта последовательность неизбежно оказывается периодической. В наиболее удачных случаях она дает n различных значений (n целых от 0 до $n - 1$), после чего возвращается к уже встретившемуся в последовательности числу, и — так как каждое из чисел имеет однозначно определенное следующее за ним число — мы повторяем уже построенную часть последовательности. Но чаще всего этим способом получаются слишком короткопериодические последовательности.

?? Головоломка 1. Периодическая последовательность.

Построим последовательность целых чисел в промежутке $(0, n - 1)$ только что описанным способом. Предположим, что n достаточно велико (например 10 000). Написать программу, определяющую период этой последовательности. Ограничение: вы не имеете права запоминать в таблице последовательные значения элементов последовательности (вы не имеете права запоминать их и в любой другой форме). Именно поэтому n предполагается достаточно большим: не может быть и речи о сохранении всех полученных значений, чтобы смотреть, встречается ли каждое новое значение среди предыдущих. Нужен другой метод. Вам предлагается обнаружить один из них...

Головоломка для маленького вундеркинда.

В прессе — как американской, так и французской — часто появляются восторженные сообщения о детях, которые обучаются работе с компьютером за несколько часов и затем объясняют своим родителям, как это делается. Разрастаются клубы по информатике, где дети пишут программы, заставляющие бледнеть профессионалов. Подающие надежды Эйнштейны информатики... Я бы ни-

когда и не предлагал следующую головоломку, если бы не был жестоко ущемлен одним из них. Понятно, что я не скажу ни где, ни когда.

Я находился в зале для практических занятий с детьми 15—16 лет. Преподаватель предложил им составить программу, бросающую две случайные кости и сообщающую число появлений каждой комбинации (см. упражнение 4). Маленький местный вундеркинд закончил свою довольно простую (не так ли!) программу, и преподаватель предложил ему следующую задачу:

пусть последовательно *n* раз выбрасывается «орел» или «решка». Сосчитать число случаев появления комбинации «орел» — «орел» — «орел», и число случаев появления комбинации «орел» — «решка» — «орел».

Мальчик очень быстро написал программу, но она не пошла. Поэтому я уселся рядом с ним и предложил ему проверить, что именно не идет. Мы вывели программу на экран. Совершенно непонятно. Плохо вложенные циклы. Поскольку ошибки были, то пришлось делать исправления вплоть до убирания одного GOTO, чтобы заменить его другим GOTO. Я сделал ему замечание, что его программа непонятна, на что он ответил, что это не имеет значения, поскольку он ее понимает. Я возразил ему, что через три месяца он сам ее не поймет: никакой реакции, это его не интересовало... Так что взаимопонимания между нами достичь не удалось. Я вижу только два возможных объяснения этого явления, в конечном счете не исключающих друг друга: многие преподаватели лицеев располагают наблюдениями такого рода.

Можно представить себе, что успешный диалог между человеком и машиной разрушительно действует на диалог между людьми. Это было бы катастрофой. Но эту столь пессимистическую гипотезу ничто реально не подкрепляет.

Гораздо более правдоподобно, что этот мальчик (как и подобные ему) встречают естественные затруднения в общении с другими людьми. Он нашел в информатике страну, в которой можно избежать такого общения и где машина принимает все, что он скажет, какова бы ни была форма и структура, лишь бы только синтаксис был соблюден. Именно поэтому феномен «информатических клубов» оказывается опасным. В рамках преподавания информатики нужно, чтобы преподаватель требовал от своих учеников анализа поставленной задачи и словесного выражения на обыденном языке результата этого анализа. Программирование может начаться только после этого. На

этом уровне преподаватель заботится о качестве стиля программ, написанных по ясному плану... (Я надеюсь, что и вы работаете именно так, иначе вас нужно остерегаться.) В обстановке, когда подростков недостаточно контролируют и они находятся как бы на самообучении, они могут оказаться свободно предоставленными своим естественным склонностям. Если организованное преподавание помогало бы им учиться выражать и структурировать свои мысли, то всетерпимость к средствам выражения, свойственная машине, позволяет им оставаться в их собственном мирке, и случайные — но настоящие — успехи ошибочно убеждают их, что искать других путей не нужно. Это ли нужно подросткам, испытывающим трудности при общении?

Вернемся к нашим баранам. Если кто-то из молодых людей и обломает на этой задаче свои зубы, то это, может быть, и правда головоломка. Пора перейти к формулировке задачи.

* Головоломка 2. Последовательности «орлов» и «решек».

Осуществим n выбрасываний «орла» и «решки» с большим n (например 10 000). Сколько раз встретится в ней данная комбинация из m следующих друг за другом выбрасываний (например, 10 раз «орел» или чередование из 10 выбрасываний «орла» и «решки», начиная с «орла»).

Есть много способов решить это упражнение. Они не все равноценны по времени вычисления. Я взял большое n и относительно большое m, чтобы прояснить явление. Ваша программа не должна работать долгие часы...

Другие азартные игры

* Игра 3. Покер — М — С.

Я не уверен, что это следует писать. Я знаю эту игру только по услышанной мною радиопередаче какой-то периферийной радиостанции (угадайте какой?). Тасуем карточную колоду. Разыгрывается некоторая сумма. Берем верхнюю карту из пачки и требуем от игрока, чтобы он угадал, является ли следующая карта младшей или старшей по отношению к только что взятой. Учитывается только число очков, а не масть карты. Валет всегда больше девяти, король больше валета, туз больше всех. Если игрок угадал правильно, сумма в игре возрастает (я не знаю точно, добавляется ли при этом некоторое фиксированное количество или сумма удваивается, но это не так

уж важно. В любом случае ваш компьютер не имеет связи с распределителем банковских билетов. Жаль, быть может...). Если он не угадывает, он теряет все. В конце некоторого фиксированного числа бросаний (кажется 6; я слушал недостаточно внимательно, я прошу прощения у упомянутой станции) игрок, если он всегда оказывался прав, присваивает сумму игры.

Составьте программу, которая позволит вам быть игроком, а компьютер пусть будет всем остальным (за исключением того, что вы называете и сумму игры). На мой взгляд, хотя я могу и ошибаться, единственная трудная задача — перетасовать карты...

?** Игра 4. Лабиринт для шахматного коня.

Лабиринты являются очень высоко ценимыми головоломками. Почему не использовать компьютер и генератор случайных чисел для построения случайных лабиринтов, которые вы затем будете пытаться пройти? Но мой микрокомпьютер не имеет графических возможностей. К тому же если у вашего такие возможности есть, то я не уверен, что желание нарисовать обычный лабиринт приводит к хорошему упражнению по программированию. Внимание часто в большей мере поглощается графическими задачами, чем более фундаментальной задачей порождения лабиринта. Тем не менее, если вам так подсказывает сердце, не стесняйтесь: стройте от случая к случаю такой лабиринт, чтобы у него был хотя бы один путь от начала к концу, и играйте с ним.

Чтобы освободиться от графических задач, рассмотрим другую форму лабиринта. Его создание составляет головоломку, а использование — игру. Пусть дана прямоугольная область, образованная n строками с p полями на каждой из них. На моем компьютере, где приходится учитывать формат экрана, числа $n = 12$ и $p = 20$ дают хорошие результаты. Занятые места считаются препятствиями (обозраченными здесь 0), пусть как-то помечены свободные места (здесь — точкой), пусть значок * обозначает всадника. Конь перемещается, как конь в шахматах: два шага в одном направлении и еще один шаг перпендикулярно предыдущему направлению. Конь может перемещаться только с одного свободного места на другое. В начальный момент он находится в правом нижнем углу. Он должен попасть в верхний левый угол (который, таким образом, тоже должен быть свободным). Число ходов игры ограничено. На рис. 1 изображен типичный пример лабиринта.

Составьте программу для компьютера для создания этого лабиринта и попытки его пройти. Так как должен существовать какой-то путь, проходящий из правого нижнего угла в правый верхний угол, то я предлагаю вам действовать следующим образом:

— возьмите случайным образом путь, связывающий эти два угла. Это — маленькая головоломка. Может быть, вы знаете задачу Эйлера о шахматном коне: составить такой путь коня по шахматной доске, чтобы он побывал на каждом поле один и только один раз. Но здесь у вас больше свободы. Тем не менее не представляется разумным проходить два раза одно и то же поле (если ваш путь будет содержать круг, то он будет предоставлять возможность для короткого замыкания, т. е. удаления этого круга). Но, может быть, это и не необходимо. Если мы много раз попадаем на одно и то же поле, то мы предоставляем много возможностей выбора, и осложняем задачу воссоздания пути. Не нужно использовать какой-либо систематический алгоритм прохода, иначе ваш лабиринт будет расшифровываться слишком быстро. Следующий за данным полем шаг на нашем пути должен выбираться случайным образом. Как тогда мы сможем быть уверены в попадании в левый верхний угол?

```
.0.0.00000000.0..000
0.0000.0.00.0...0.00
0.0.00000.000.0..000
0000000.000000..0000
000.000000.0.0000.00
000000000000.0..0000
0000.000000.00000000
00000000000000.00000
0000000000000000.000
00000000.0000000.000
0.00.0000000000.0.00
.0000000000.0000000*
```

Рис. 1

— получив однажды такой путь, отметьте его. Затем вы случайным образом распределяете препятствия на полях, не принадлежащих выбранному пути. Степень заполнения этих полей является параметром, который вы подберете по опыту. Если вы поставите слишком мало препятствий, ваша шахматная доска будет почти пустой, и будет много возможных путей, так что лабиринт не получится. Если же вы поставите много препятствий, то

путь будет почти полностью определен (на рисунке препятствия занимают приблизительно 2/3 полей. Это — верхняя грань);

— когда это сделано, вы снимаете обозначения полей выбранного пути, заменяя их точками. Лабиринт готов к показу.

Остается обеспечить движение коня. Вот как действую я. Сначала я подсчитываю число полей на исходном пути, которые были выбраны случайно, и вывожу это число в качестве верхней границы числа ходов. Я свидетельствую, что всегда обнаруживался более короткий путь. Я не пытался объяснить этот экспериментальный факт...

Компьютер сообщает число оставшихся ходов и требует ваших указаний о движении. Ответ дается в виде двух букв: первая из этих букв дает направление, в котором нужно переместиться на два шага, вторая буква дает перпендикулярное предыдущему направление, в котором нужно сделать один шаг: Н — для нижней, В — для верхней, П — для правой, Л — для левой сторон. В случае на рис. 1 первое движение предписывает ЛВ — два шага влево, один вверх.

Компьютер анализирует ответ. Если превышено число ходов или ход встречает препятствие, то игрок проигрывает. Если нет — звездочка, изображающая коня, перемещается в новое положение, число оставшихся ходов уменьшается на единицу, и игра продолжается.

Игра 5. Спящая красавица.

Краткое содержание предыдущих эпизодов. Доктор Жабуэ не убил великолепную Жюли, он только приостановил жизненные процессы. Ее мог бы разбудить надлежащий лицевой массаж, но это его не беспокоит, впереди еще много времени. Из замка вывезено все, что имело хоть какую-то ценность: обстановка, картины, произведения искусства... Молодой повеса обнаруживает пустой замок и находит, что он должен быть замечательным треком для мотогонок...

13-й эпизод.

Рыжий Тони входит в темную комнату. Несмотря на грохот мотоцикла, отчетливо воспринимается равномерный храп. Он зажигает фару и обнаруживает безмятежно спящую прекрасную Жюли. Ослепленный ее красотой, он приближается к ней и гладит ее по лицу. Ничего больше и не нужно. Жюли внезапно просыпается и, приходя в сознание с удивительной быстротой, восклицает: «Бежим отсюда скорее, в замке западня, все сейчас взо-

рвется». Тони садится на мотоцикл. Жюли вскакивает на сиденье за Тони и пристегивается к нему. Но уже повсюду гремят взрывы, и огонь охватывает деревянный потолок. Тони мчится зигзагами среди обломков. Обрушиваются куски горящих балок, угрожая раздавить их в любой момент.

Удастся ли им выбраться из замка? Продолжение в следующем эпизоде.

Эта игра является вариантом предыдущей. Вы представляете замок тем же самым прямоугольным пространством, где точки обозначают свободные места, нули — препятствия, а звездочка сообщает местоположение Тони. В начале игры Тони находится в правом нижнем углу, а выход находится в левом верхнем углу. Препятствий вначале крайне мало. После каждого хода Тони компьютер случайным образом формирует новые препятствия и размещает их на игровом поле. Если Тони оказывается на месте одного из них, то он раздавлен и для него все кончено...

Чтобы оставить Тони хоть какой-то шанс, я предпочитаю устанавливать препятствия парами — поочередно вертикальными и горизонтальными. После небольшого наблюдения можно заметить, что движение большей частью идет в тесных коридорах, куда препятствия не могут попасть (нужно по крайней мере два смежных свободных поля, чтобы там могло разместиться препятствие).

Если не принимать мер предосторожности, то препятствия могут полностью загородить путь к выходу. Может быть, предпочтительнее условиться, чтобы хоть какой-то путь оставался свободным. Каким образом — на ваше усмотрение. В своей первой версии такой программы я этого не сделал. Чаще всего выход оказывался блокированным и игра могла быть выиграна лишь в исключительных случаях.

2. ИГРЫ С ЧИСЛАМИ

Арифметические развлечения

Есть много примеров арифметических игр, головоломок и развлечений. Их можно найти в [BAL], [BER], [KUE]. Мы обращаемся и к другим источникам и добавляем некоторые задачи, которые представляют интерес собственно с точки зрения программирования. Многие арифметические головоломки можно сделать вручную: для чего составлять программу? Очень часто вам нужно сделать

часть работы на руках, а машина сделает остальное. Это вы должны правильно распределить работу, иначе время вычисления может оказаться чрезмерным. Строгих правил здесь нет. Одна и та же задача может иметь много решений, и методы решения одной и той же задачи могут быть различными. Это и создает игровую ситуацию.

Группировка различных головоломок по темам более или менее условна. Начнем с наиболее простых задач.

ДЛЯ РАЗМИНКИ.

Головоломка 3. Вращающееся число.

Найти такое число, оканчивающееся на 5, что, умножая его на 5, мы получим новое число, полученное из предыдущего вычеркиванием цифры 5 на конце и приписыванием ее в начале.

Это легко...

Та же задача с заменой 5 на 2.

Можно ли заменить здесь 5 какой-нибудь цифрой, отличной от 0?

** Головоломка 4. Квадратный корень.

Извлечь целый квадратный корень с недостатком из очень длинного целого числа (намного более длинного, чем наибольшее целое, которое воспринимается вашим компьютером, например, содержащего 50 или 100 значащих цифр).

Числовые последовательности

Вот две известные в информатике головоломки. Сожалею, что обманываю ожидания своих коллег, которые не найдут здесь ничего нового...

?* Головоломка 5. Последовательность Хэмминга.

Рассмотрим числа, не имеющие других простых делителей, кроме 2, 3 и 5. Расположим их в возрастающем порядке. Это и есть последовательность Хэмминга. Вот ее начало:

2 3 4 5 6 8 10 12 15 16 18 20 24 25 27 30 32 36 40 45 48 50...

Составьте программу, выписывающую n первых членов этой последовательности для большого n. Внимание: вы должны порождать последовательность Хэмминга в порядке возрастания ее членов. Нетрудно, например, взять степени тройки и разместить их в последовательность. Вы же образуйте последовательность от номера 1 до номера $i - 1$, а затем вычислите и поставьте на место

элемент последовательности с номером i. В этом-то и головоломка...

?** Головоломка 6. Счастливые числа.

Унтер-офицер собирает своих людей, чтобы решить, кого отправить в наряд на картошку.

«Постройтесь гуськом и рассчитайтесь, начиная с 2». Первый из стоящих говорит 2, следующий — 3, следующий — 4 и т. д.

«Первый в ряду, выйди из строя. Ты освобожден от наряда. Какой у тебя номер?»

«Второй»,— отвечает солдат.

«Начиная со второго, рассчитаться по два; тем, кому не выпадет 2, выйти из строя; они пойдут в наряд».

И процесс возобновляется. Первый из вышедших из строя имеет номер 3, и он счастлив: он освобожден от наряда. Теперь рассчитываются по трое, начиная с 3 — с того, кто первым вышел из строя за нарядом...

Составьте программу, выписывающую n первых счастливых чисел для большого n (100, даже 500). Внимание: в чем состоит головоломка: каждый член последовательности должен вычисляться, исходя из данных значений предыдущих счастливых чисел. У вас есть i первых, вычислите следующее. В таблице-то легко вычеркивать... Вот первые счастливые числа:

2 3 5 7 11 13 17 23 25 29

Счастливые числа — не обязательно простые, а простые числа — не обязательно счастливые...

??? Головоломка 7. Дьявольская последовательность.

Марк Твен описал в своих рассказах жуткую историю. Человек прочел глупые стихи вроде

Кондуктор, отправляясь в путь,
Не рви билеты как-нибудь,
Стриги как можно осторожней,
Чтоб видел пассажир дорожный:

Синий стоит восемь центов,
Желтый стоит девять центов,
Красный стоит только три.
Осторожней режь, смотри!

Припев:

Режьте, братцы, режьте! Режьте осторожно!
Режьте, чтобы видел пассажир дорожный!

(Я цитирую по памяти, но дух соблюден.) Он был порабощен ритмом этих стихов, что стало настоящим наважде-

нием. Если он начинал писать, его перо выводило «Режьте, братцы, режьте». Если он встречал кого-нибудь, он не здоровался с ним, а говорил «Режьте, братцы».

Он пробовал управлять собой, но это подрывало его здоровье. Он решил обратиться к своему священнику и объяснить ему, в чем дело, и читал ему это маленькое стихотворение, подчеркивая его ритм, пока пастор не выучил его наизусть. Ушел он исцеленный.

Но в воскресенье пастор начал проповедь словами «Режьте, братцы, режьте». Что бы ни было в гимне, который он запевал, слова были одни — «Режьте, братцы, режьте...» Его жизнь стала адом. Он не мог исцелиться, пока в один прекрасный день ему не удалось злодейски обучить этому стихотворению одного профессора университета...

Нижеследующее и есть «режьте, братцы, режьте». Оно преследует меня долгие годы. Я потерял массу времени на размышления о нем без сколько-нибудь значительного успеха. Но ничто меня не занимает в большей степени. Моя единственная надежда освободиться от него — это то, что вы им заинтересуетесь...

Последовательность определяется следующим образом: первый член этой последовательности есть произвольное нечетное число, отличное от единицы. Следующее за числом p равно

$p/2$, если p четно,

$3p+1$, если p нечетно.

Последовательность заканчивается, когда в ней встречается значение 1.

Вот последовательность, которую мы получим, исходя из 7:

7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Нет никакой надежды, что вам удастся доказать, что для любого нечетного числа в качестве начального значения последовательность достигает единицы.

Но в высшей степени увлекательно составить эту крошечную программу и посмотреть, как она работает. Испытайте число 27 в качестве начального значения: вы получите очень длинную последовательность, среди элементов которой есть 9232. Если вы изучите ряды чисел, получаемые для начальных значений, взятых среди нечетных целых от 3 до 99, вы получите довольно много патологических последовательностей, не всегда сильно отличающих-

ся. Все это очень смущает. Ни один специалист по теории чисел еще не смог доказать, что такая последовательность принимает значение 1 для любого начального значения. Не больше известно и о том, почему некоторые из этих последовательностей — короткие, а другие — слишком длинные...

Эта программа замечательно иллюстрирует то, что называется «проблемой остановки». Существуют простейшие программы, относительно которых нет уверенности, что они остановятся...

Теперь, когда вы уже познакомились с этой последовательностью, получите предмет головоломки. Заметим сначала, что если p нечетно, то мы переходим к $3p + 1$ — числу, отличному от 1. Очевидно, что непосредственно предшествующий шаг есть деление на 2. Поэтому можно изменить правило построения последовательности описанным ниже образом: следующее за числом p равно

$p/2$, если p четно,
$(3p + 1)/2$, если p нечетно.

Это вычеркивает некоторые члены предыдущей последовательности, не меняя проблемы остановки:

7 11 17 26 13 90 10 5 8 4 2 1

Вы можете пойти еще дальше в том же направлении, объединяя вместе все последовательные шаги, действующие по правилу $(3p + 1)/2$, и все следующие за ними шаги, состоящие в делении на два. Вы получите два новых правила перехода, гораздо более уплотненные. Свяжите их и пустите в ход. Для числа 7 вы должны без задержки получить последовательность

7 13 5 1

Это позволяет рассматривать обобщения задачи. Пусть k — нечетное число. Возьмем в качестве правил перехода следующие:

$p/2$, если p четно,
$k * p + k - 2$, если p нечетно.

Возможно уплотнение, аналогичное предыдущему. Для $k = 5$ следующее за числом 3 есть 3, и существуют исходные точки, для которых программа не останавливается. Для $k = 7$ она идет точно так же. Так что проблема остановки связана со свойством числа k. Я бы здесь... Впрочем, мало ли чего я хочу!

Зашифрованные операции

Это — класс самых разнообразных задач. Задаются точные арифметические операции, в которых некоторые цифры либо стерты, либо заменены буквами. В данной операции одна и та же буква всегда заменяет одну и ту же цифру, и разные буквы представляют поэтому разные цифры. Нужно восстановить исходную операцию. Есть случаи, в которых это сводится к решению системы уравнений с неизвестными, представляющими собой букву,— системы, решение которой дает также решение исходной задачи. Компьютер не видит ничего скрытного. Таким образом, если что-то не так, то нужно действовать систематически методом проб и ошибок. Нужно выбрать значения для одних букв и получить с их помощью значения остальных. Нужно проверить, что разным буквам соответствуют разные значения. После конечного числа попыток мы получим решение — если оно единственно — или список всех возможных решений. А еще существуют промежуточные решения: вычисление ограничивает число осуществляемых попыток.

Головоломка 8. SEND MORE MONEY. *)

Это — лаконичная телеграмма английского студента своему отцу. История умалчивает о том, как отец это принял и были ли отправлены деньги...

$$\begin{array}{r} \text{SEND} \\ +\ \text{MORE} \\ \hline \text{MONEY} \end{array}$$

Программа очень легкая. Время вычисления короткое. Едва ли это головоломка. Как раз для тренировки...

Головоломка 9. HELP THE YOUNG. **)

Конечно, конечно. Почему бы не послать им еще денег? Та же задача:

HELP + THE = YOUNG

Отметим разницу с предыдущей задачей. Предыдущая использовала не все цифры от 0 до 9. В этой участвуют все. Можете ли вы воспользоваться этим?

DEVOIR, LEÇON, ÉLÈVE. **)

---

*) «Пришлите побольше денег.»

**) «Помогите молодому человеку.»

***) «Нужно, лекция, ученик.»

Есть аналогичные зашифрованные сложения по-французски. Например, такая:

ÉLÈVE + LEÇON = DEVOIR

? Головоломка 10. Зашифрованное умножение.

Довольно сложений, это становится скучным. Вот зашифрованное умножение:

ABCDE * 9 = FGHIJ

Здесь 10 букв представляют 10 различных цифр, так что одна из них равна 9. Можно сразу кое-что сказать о возможных значениях букв, но чтобы получить решение, придется идти буквально ощупью. Столько же придется искать и компьютеру.

?* Головоломка 11. Забавное число.

Число 123456789 обладает забавными свойствами:

123456789 * 2 = 246913578

Как и исходное, удвоенное число образовано всеми девятью цифрами, кроме 0.

123456789 * 4 = 493827156

Результат снова образован девятью цифрами, отличными от 0.

123456789 * 5 = 617283945

По-прежнему 9 цифр.

123456789 * 7 = 864197523

Опять 9 цифр, и это еще не все.

123456789 * 8 = 987654312

Но это не работает ни для 3, ни для 6. Это не может работать и для 9, потому что в результате больше 9 цифр.

Тем не менее есть много чисел, образованных всеми 9 цифрами (кроме 0), которые после умножения на 3 дают результат, образованный теми же девятью цифрами. Можете ли вы дать список всех таких чисел, оканчивающихся на 9? И также список тех, которые кончаются на 3?

Можно ли распространить использованный метод на случай умножения на 6?

Доказательства теорем

Компьютер можно использовать для доказательства теорем. Это — трудная задача искусственного интеллекта. Мы снабжаем компьютер правилами вывода, даем формулировку того, что требуется доказать, и исходные аксиомы. Компьютер пытается найти последовательность правил вывода, которые могут привести от исходных данных к требуемым результатам.

Здесь обо всем этом речь не идет. Я предлагаю вам только взглянуть на путь, использованный для доказательства с помощью компьютера знаменитой проблемы четырех красок: любая географическая карта может быть раскрашена четырьмя красками так, что любые две территории, имеющие общую границу, раскрашены разными красками. Общая идея состоит в том, чтобы доказать вручную или, в случае необходимости, с помощью программы, что проблема будет решена полностью, если будет известно ее решение в некотором конечном числе случаев. Эти случаи исследуются на компьютере. Вот примеры, доступные этому методу.

Внимание: вы должны бороться с проблемой сложности. Если вы не будете принимать никаких мер предосторожности, число подлежащих исследованию случаев может оказаться невероятно большим и работа компьютера станет невозможной: ведь перед вами не вечность... Равновесие между подготовительной работой (доказательством малых теорем) и работой компьютера оценивается в зависимости от ваших возможностей, одновременно в области математических доказательств и в ресурсах вашего микрокомпьютера. К сожалению, не говорите: эта программа отнимает уйму времени, я перепишу ее на ассемблере. Это — худшее из решений. Все, что я вам предлагаю, осуществимо на Бейсике за разумное время. Если ваша программа требует уйму времени, значит, она плохо придумана.

Головоломка 12. Теорема 153.

Этот пример заимствован из [MJB]. Образуем числовую последовательность следующим образом:

— начальный элемент — произвольное натуральное число, кратное трем,

— за любым элементом последовательности следует число, равное сумме кубов всех цифр данного элемента.

Теорема. Любая такая последовательность становится (начиная с некоторого места) постоянной, равной 153.

Пример. Начнем с 33:

$$33$$
$$3^3 + 3^3 = 54$$
$$5^3 + 4^3 = 189$$
$$1^3 + 8^3 + 9^3 = 1242$$
$$1^3 + 2^3 + 4^3 + 2^3 = 81$$
$$8^3 + 1^3 = 153$$
$$1^3 + 5^3 + 3^3 = 153$$
$$1^3 + 5^3 + 3^3 = 153$$

и теперь последовательность стала постоянной.

Используйте ваш компьютер для доказательства этой теоремы.

? Головоломка 13. Варианты.

• Нелегко сказать, какую роль в предыдущей теореме играет то, что исходное число кратно трем. Но от вас не потребует чрезмерных усилий в общем случае, что два последовательных числа последовательности имеют равные остатки при делении их на 3. В последовательностях, которые мы стали изучать, все члены последовательности делятся на 3. Можно доказать также, что все члены последовательности, кроме, быть может, первого, делятся на 9.

Если взять натуральное число, не кратное трем, то все члены соответствующей последовательности будут иметь один и тот же остаток при делении на 3. Что, кроме этого, вы можете узнать о поведении этих последовательностей?

Если при переходе к следующему члену последовательности вы будете брать сумму квадратов цифр (вместо того, чтобы брать сумму кубов), то все будет не намного лучше. Можете ли вы доказать следующую теорему: каково бы ни было натуральное число, взятое в качестве первого элемента последовательности, эта последовательность содержит число, не превосходящее 4?

? Головоломка 14. Теорема 6174.

Построим последовательность натуральных чисел следующим образом. Начальный элемент — натуральное число с четырьмя цифрами, которые не все равны между собой. Мы переходим от данного члена последовательности к следующему по такому правилу.

Пусть a, b, c, d — четыре цифры, представляющие десятичную запись данного числа. Расположим их в порядке убывания слева направо и получим первое число. Расположим их в обратном порядке и вычтем это второе число из первого. Это и есть искомый следующий член последовательности.

Теорема. Эта последовательность для любого начального элемента становится (начиная с некоторого места) постоянной, равной 6174.

Пример. Начнем с 7815:

8751 — 1578 = 7173
7731 — 1377 = 6354
6543 — 3456 = 3087
8730 — 0378 = 8352
8532 — 2385 = 6174
6174 — 1467 = 6174

Используйте ваш компьютер для доказательства этой теоремы. Это окажется намного проще, чем в предыдущей головоломке, поскольку имеется всего лишь 9000 чисел с четырьмя цифрами, и нужно исследовать 9000 последовательностей. Но вы можете сделать число испытаний намного меньше этого...

?? Головоломка 15. Господин S и господин P *).

Вот одна из наиболее классических арифметических головоломок. Выберем два натуральных числа, больших единицы, но меньших ста. Значение их суммы сообщено господину S, значение их произведения — господину P. Ни один из них не знает, какое число сообщено другому. Господин P звонит господину S по телефону.

P. Я не могу найти эти два числа.

S. Я знаю, что вам это и не удалось бы.

P. Ах, так... Но тогда я их знаю!

S. Ну, тогда и я тоже их знаю!

Рассуждение позволяет существенно видоизменить задачу, и даже более того — предъявить решение. Много ли их? Используйте ваш компьютер, чтобы их найти.

Простые числа

??** Головоломка 16. Чемпион головоломок. На мой взгляд, наиболее замечательная арифметическая головоломка, над которой мне пришлось особенно долго работать и которая дала мне возможность получить некоторые удовлетворительные результаты,— это, конечно, проблема простых чисел. Пусть дано число n (конечно, нечетное) и достаточно большое; сказать, является ли оно простым и, если можно, дать его разложение на простые множители.

*) S — первая буква слова «somme» (фр. сумма), P — слова «produit» (фр. произведение).— *Примеч. ред.*

Если не предполагать, что n велико, то есть простой способ действовать: делить n на простые числа и смотреть, удается ли деление без остатка. Если да, то число составное и допускает разложение в произведение. Впрочем, при таком методе многие делители можно вообще не рассматривать. Если n есть произведение двух сомножителей p и q:

$$n = p * q,$$

то либо $p = q$, либо один из сомножителей больше другого, так что можно считать, что p — делитель, q — частное и $p \leqslant q$. Поэтому будем делить n на последовательно возрастающие простые числа, для которых частное больше или равно делителю. Так как мы не располагаем таблицей простых чисел, то используем последовательность делителей, которая заведомо содержит все простые числа, например, последовательность нечетных чисел или лучше целых чисел вида $6k \pm 1$.

Число операций растет как квадратный корень из n. Если вы добавите к n одну цифру, то вы увеличите время вычисления примерно раза в три. Но более важно другое. Если вы увеличиваете n, вы можете превысить «арифметические способности» своего компьютера. Как вы узнаете, правильно ли выполнено деление? Предел, которого вы можете достичь таким образом, существенно зависит от марки вашего микрокомпьютера *).

Таким образом, вы должны бороться со следующими трудностями:

— точность вашего компьютера. Вам нужно иметь возможность делать вычисления с повышенной точностью, а это очень дорогостояще по времени;

— число требуемых операций;

— доверие к вашей пр грамме. Если ваша машина сообщает вам, что

$$9873564383 = 631181 * 15643,$$

то вы, вероятно, сможете проверить этот результат на вашем микрокалькуляторе. А если компьютер сообщит вам, что 9873564401 — простое число, то как вы это проверите? Проделав вычисления на руках?

Вот основы метода Ж.—М. Полларда [POL].

По данному числу n (нечетному натуральному) строится последовательность по описанному ниже правилу:

*) Да и от языка, который вы используете.— *Примеч. ред.*

— первый член последовательности равен 2;

— следующий за x элемент равен $x^2 - 1$ по модулю n (остатку от деления $x^2 - 1$ на n).

Оказывается, что эта последовательность периодична. Это легко видеть. Остаток от деления на n есть неотрицательное целое, меньшее n, поэтому не может быть более n различных остатков. Поэтому неизбежно, что как только число членов превысит n, среди членов последовательности мы получим два одинаковых, что и означает периодичность последовательности. Но она может оказаться периодической с намного более коротким периодом, чем n. Вот, например, последовательность для $n = 137$:

$a_1 = 2$ $a_2 = 3$ $a_3 = 8$ $a_4 = 63$ $a_5 = 132$
$a_6 = 24$ $a_7 = 27$ $a_8 = 43$ $a_9 = 67$ $a_{10} = 104$
$a_{11} = 129$ $a_{12} = 63 = a_4$

Последовательность периодична с периодом 8.

Пусть дана последовательность, вычисленная для некоторого n. Предположим, что n делится на s, и что соответствующая числу s последовательность периодична с периодом p.

Для достаточно большого i имеем $a_{i+p} = a_i$ по модулю p, следовательно, $a_{i+p} - a_i$ делится на p. Так как, кроме того, и n делится на p, то наибольший общий делитель (НОД) чисел $a_{i+p} - a_i$ и n отличен от 1 *).

Построим последовательность Полларда для $n = 22879$:

$a_1 = 2$ $a_2 = 3$ $a_3 = 8$ $a_4 = 63$ $a_5 = 3968$
$a_6 = 4271$ $a_7 = 6877$ $a_8 = 2235$ $a_9 = 7602$
$a_{10} = 20928$ $a_{11} = 8486$ $a_{12} = 11982$

НОД чисел $a_{12} - a_4$ и $n = 22879$ есть 137, делитель числа n.

Если мы способны сказать, становится ли данная последовательность периодической (головоломка 1), то мы располагаем быстрым методом определения, имеет ли

*) Повторим эти рассуждения чуть более подробно. Пусть

$$a_1 = 2, \quad a_{i+1} = a_i^2 - 1 \bmod n,$$

$$b_1 = 2, \quad b_{i+1} = b_i^2 \bmod s$$

— последовательности, соответствующие числам n и s соответственно. Тогда легко доказать по индукции, что $b_i = a_i \bmod s$. Одним из периодов последовательности $\{a_i\}$ является n. Значит, n является периодом и для последовательности $\{b_i\}$. Известно, что любой период последовательности кратен ее минимальному периоду. Так как p, по определению, является минимальным периодом последовательности b_i, то n делится на p. — *Примеч. ред.*

данное число делитель. Можете играть. Это не такая уж простая программа...

Есть тест на простоту числа, основанный на так называемой малой теореме Ферма: если n — простое, причем число n не является делителем a, то

$a^{n-1} = 1$ по модулю n.

Представим n в виде $n = 2^s m + 1$. Назовем число n сильно псевдопростым по основанию a, если выполнено одно из следующих двух условий:

либо $a^m = 1$ по модулю n,

либо $a^{m2^r} = n - 1$ по модулю $n = 2^s m + 1$ для некоторого r, $0 \leqslant r < s$.

Очень мало сильно псевдопростых чисел, не являющихся простыми; так

$2047 = 23 * 89$ — сильно псевдопросто по основанию 2,

$1373653 = 829 * 1657$ — по основанию 2 и 3,

$25326001 = 2251 * 11251$ — по основанию 2, 3 и 5,

$3215031751 = 151 * 751 * 28351$ — по основанию 2, 3, 5 и 7.

Метод интересен, потому что a^n вычисляется за время, растущее не быстрее, чем $\ln n$. Это утверждение вытекает из соотношений:

$$a^0 = 1,\ a^1 = a,$$
$$a^{2n} = (a * a)^n,\ a^{2n+1} = (a * a)^n * a.$$

Все, что нужно для работы, у вас есть. Больше делать нечего, кроме собственно составления программы.

Кстати: знаете ли вы две универсальные конструкции в информатике? Первая — «известно, что...». Вторая — «это и нужно сделать...».

Таинственные программы

Я надеялся не приводить в этой книге никаких готовых программ. Программирую не я, а вы. И я не очень люблю смотреть, как подростки копируют программу, набирая ее на клавиатуре и при этом не отдавая себе отчета в том, что она делает и как устроена. Но сказать, что делает та или иная программа, может оказаться настоящей головоломкой. Программы, которые мы будем обсуждать, на-

писаны на некотором воображаемом языке*). Вам придется по крайней мере сделать усилие, чтобы перевести их на ваш обычный язык: Бейсик, LSE или Паскаль.

Условная команда записывается в виде

ЕСЛИ условие ТО последовательность команд
КОНЕЦ_ЕСЛИ

(последовательность команд выполняется тогда и только тогда, когда условие истинно)
или

ЕСЛИ условие ТО последовательность команд
ИНАЧЕ последовательность команд
КОНЕЦ_ЕСЛИ

(если условие истинно, то выполняется последовательность команд, заключенная между ТО и ИНАЧЕ, в противном случае выполняется та последовательность команд, которая расположена между ИНАЧЕ и КОНЕЦ_ЕСЛИ).

В обоих случаях КОНЕЦ_ЕСЛИ играет роль закрывающей скобки, связанной с открывающей скобкой ЕСЛИ.

Мы будем использовать цикл

ПОКА условие ВЫПОЛНЯТЬ
последовательность команд
ВЕРНУТЬСЯ

Последовательность команд, содержащаяся между ВЫПОЛНЯТЬ и ВЕРНУТЬСЯ, повторяется, ПОКА условие истинно.

* Головоломка 17. Для забавы.

Вот легко понимаемая программа. Здесь n и b — два натуральных числа и b нечетно (это существенно)

```
ПРОЧЕСТЬ n, b
ПОКА n ⩾ b ВЫПОЛНЯТЬ
   ЕСЛИ n четно ТО n := n/2
   ИНАЧЕ n := n − b
   КОНЕЦ_ЕСЛИ
ВЕРНУТЬСЯ
СООБЩИТЬ ЕСЛИ n = 0 ТО'ДА'
ИНАЧЕ 'НЕТ' КОНЕЦ_ЕСЛИ
```

Вы можете попробовать выполнить ее вручную для

$n = 2^{77} - 3$, $b = 7$.

---

*) Этот язык описан на стр. 7—8 выше. Здесь лишь кратко напоминаются формы записи условных операторов и операторов цикла.— *Примеч. ред.*

Забавно, не правда ли? Несмотря на свою исключительную простоту, эта программа, кажется, новая...

*** Головоломка 18. Посерьезнее.

Эта — несомненно более трудная. И тоже неопубликованная. Боюсь, что вы можете избаловаться... На вход программы подается n — нечетное натуральное число.

```
ПРОЧЕСТЬ n
q := (n — 1)/4; p := целая_часть (q)
ЕСЛИ q ≠ p ТО СООБЩИТЬ 'НЕТ';
КОНЕЦ_РАБОТЫ КОНЕЦ_ЕСЛИ
ЕСЛИ нечетное p ТО СООБЩИТЬ 'НЕТ';
КОНЕЦ_РАБОТЫ КОНЕЦ_ЕСЛИ
a := 4; b := 1

ПОКА p ⩾ a ВЫПОЛНЯТЬ
p := p/2
ЕСЛИ нечетное p ТО p := p — a/2 — b;
b := a — b КОНЕЦ_ЕСЛИ a := a + a
ВЕРНУТЬСЯ
ЕСЛИ p = 0 ТО СООБЩИТЬ b;
КОНЕЦ_РАБОТЫ КОНЕЦ_ЕСЛИ
ЕСЛИ p + 2 * b = a ТО СООБЩИТЬ a — b;
КОНЕЦ_РАБОТЫ КОНЕЦ_ЕСЛИ
ЕСЛИ p = 4 * (a — b) ТО СООБЩИТЬ 2 * a — b;
КОНЕЦ_РАБОТЫ КОНЕЦ_ЕСЛИ
СООБЩИТЬ 'НЕТ'; КОНЕЦ_РАБОТЫ
```

Я не запрещаю вам перевести эту программу на ваш любимый язык, а затем испытать ее для различных значений n. Есть маленький шанс, что вы угадаете, на что она способна. Это не очевидно!

** Головоломка 19. Вклад Жака Гебенстрейта.

Я обязан Жаку Гебенстрейту следующей программой. Она была предложена в том виде, в каком я ее привожу, без какого-либо комментария (это было сделано без злого умысла с его стороны: сам он получил не больше от того, кто дал ему эту программу).

```
ПРОЧЕСТЬ a, b; p := max (a, b); q := min (a, b)
  ПОКА q ⩾ eps ВЫПОЛНЯТЬ
  r := (q/p)²; s := r/(r + 4)
  p := (2 * s + 1) * p;  q := s * q
ВЕРНУТЬСЯ
результат := p
```

Как вам кажется, что вычисляет эта программа?

3. ИГРЫ БЕЗ СТРАТЕГИИ

Общие предложения

Мы собираемся предложить здесь игры для программирования. Мы выбрали их потому, что они не требуют придумывания выигрывающей стратегии при составлении программы. Каждая игра ставит вас перед, вообще говоря, непредсказуемой ситуацией. Вы должны играть, соблюдая правила и имея в виду добиться определенной цели. Вам следовало бы развить собственную стратегию. После того, как вы предложили свой ход, компьютер сообщает новое состояние игры, затем изменяет это состояние некоторым почти полностью определенным образом. Ваша стратегия должна оценить, как именно.

Программирование этих игр не представляет заметных трудностей. Это — не головоломка (если исключить лабиринт для коня, который представляет настоящую трудность). Но нужно работать очень тщательно, чтобы компьютер соблюдал правила и не жульничал. Ошибки программирования всегда достойны осуждения. Здесь может возникнуть искушение сделать все быстро, потому что в конце концов всегда что-то получится. Тщательно изучите задачу, продумайте прежде чем писать, составьте план вашей программы, разберите подзадачи отдельно...

И г р а 6. Гениальный ответчик.

Я не думаю, что есть еще хоть кто-нибудь, кто не знает игру, которую М. Мейрович назвал «гениальный ответчик» *). Обычно она играется с цветными шашками (6 и 8 различных цветов в зависимости от изготовления). Играющий должен угадать сделанную из этих шашек тайную комбинацию. Так, в варианте «мини» вы должны угадать комбинацию из четырех шашек, например: ГОЛУБАЯ ЖЕЛТАЯ ЖЕЛТАЯ КРАСНАЯ (в этом порядке). Второй игрок — ведущий. Это он выбирает комбинацию для угадывания и он оценивает ходы, которые вы делаете. Ваш ход в игре состоит в том, что вы предлагаете комбинацию, содержащую то же число шашек, из того же набора цветов, например, КРАСНАЯ ГОЛУБАЯ ЖЕЛТАЯ ГОЛУБАЯ.

Ведущий сообщает вам, сколько шашек из вашей комбинации содержат правильный цвет на правильном месте: здесь — 1, третья шашка ЖЕЛТАЯ, как и в неизвестной вам комбинации. Затем он сообщает вам, сколько шашек имеют правильный цвет, но стоят на неправильных местах. Здесь 1, так как вы предложили красную шашку, но она

*) В оригинале «master-mind». — *Примеч. ред.*

на неверном месте. Согласно традиции, ведущий выставляет столько черных шашек, сколько в вашем ответе шашек правильного цвета на правильном месте, а затем столько белых шашек, сколько шашек правильного цвета на неверных местах.

Составьте программу, заставляющую ваш компьютер играть роль ведущего. Он дожен случайным образом выбирать комбинацию и, конечно, не сообщать ее. Когда вы предлагаете свою комбинацию, компьютер должен также сообщить вам, сколько черных и белых шашек она заслуживает. Если в конце обусловленного заранее числа попыток вы не достигли результата, компьютер вас не поздравит, но сообщит загаданную комбинацию.

Введите в вашу программу параметры по числу цветов и числу шашек в комбинации, как и по степени сложности:

простая: 4 шашки, 6 цветов, 6 попыток;
средняя: 6 шашек, 8 цветов, 8 попыток.

Особой трудности нет. Игра заведомо приятна. Отладьте диалог...

Игра 7. Пляж Ботафого.

Несколько лет назад я отправился вести курс в Понтификальном университете Рио-де-Жанейро. Часть моей семьи смогла присоединиться ко мне на несколько дней. Мы отправились посмотреть на знаменитую «Сахарную голову», расположенную вблизи Ботафого. Мы прибыли на место. В нескольких стах метров перед нами «Сахарная голова» на берегу маленькой бухты открывала весьма привлекательный пляж. Чтобы его достичь, мы должны были преодолеть одно препятствие: автостраду. Нам очень хотелось отправиться на пляж и насладиться морем (мы еще не знали, насколько оно может быть загрязнено!). Но очевидным образом не было никаких средств пересечь эту автостраду. Ни один переход не пересекал ее поверху, это мы видели. Не было поблизости и подземных переходов. Мы не говорили по-бразильски (он произошел от португальского, но он отошел от языка, на котором говорят в Лиссабоне, так же сильно, как язык Квебека отличается от французского в Париже). Наконец, мы попытались что-нибудь понять с помощью моего приблизительного английского. «Чтобы попасть туда? Пересеките автостраду...» Это не было многообещающим развлечением. Движение было интенсивным. Бразильцы водят машины на большой скорости. Не слушая ничего, кроме зова нашей отваги, мы успешно достигли покрытой дерном полосы,

разделяющей два направления автострады, и впали в глубокое уныние. Никогда нам не добраться до цели! Но отступать было некуда. Либо в одном, либо в другом направлении, но дорогу нужно было пересечь. Я не знаю, каким образом нам удалось ее перейти. И вот, после того, как мы решили, что пробил наш последний час, мы вылезли из моря на пляж Ботафого и — обнаружили в ста метрах подземный переход, позволявший безопасно перейти дорогу. Когда я позже рассказывал эти злоключения одному из своих коллег, он заявил мне, что вообще молодые люди не имеют привычки пользоваться ни подземными, ни наземными переходами, они бросаются прямо под автомобили безо всякой боязни. Он рассказал мне также, что после торжественного открытия надземного перехода, который пересекал ту же автостраду немного выше Фламенко, в одном журнале был опубликован юмористический рисунок. На нем была изображена мать семейства, катящая детскую коляску и тянущая другой рукой ребенка, чтобы перейти пешком автотрассу у надземного перехода, говоря «Какой удобный переход! Наконец-то мы сможем переходить дорогу в тенечке...»

Все это навело меня на вот какую игру. На экран выводится рисунок, символизирующий автостраду с n полосами движения. По каждой полосе движутся автомобили. В начале игры вы находитесь на краю автострады. Вы можете либо оставаться там, где вы есть, либо прыгнуть на шаг вперед. Автомобили перемещаются согласно неизвестному закону. Если один из них достигает занимаемого вами положения или проходит по нему, то вы раздавлены. А если нет, то ваш ход. Вы снова можете либо остаться неподвижным, либо продвинуться на шаг вперед, либо вернуться на шаг назад, и цикл возобновляется. Вы выигрываете, если вы достигаете другой стороны дороги, не будучи раздавленным.

Вот несколько предложений по реализации игры. Я представляю автостраду следующим образом.

Расстояние между машинами постоянно. В верхней полосе оно равно 18 (17 точек между двумя машинами) и возрастает на 1 в каждой следующей полосе (24 точки между двумя стрелками в нижней полосе). Стрелки представляют машины острием в направлении их перемещения. Тире указывают место вашего перехода, но это отнюдь не переход «зебра»: ни одна машина не замедлит хода, чтобы дать вам перейти! В начале игры вы находитесь вне автострады, как на рис. 2.

Скорость машин постоянна. В верхней полосе я выбрал 5: любое перемещение машин продвигает их на 5 точек влево. В результате одна из машин может уйти влево или справа может появиться новое транспортное средство, если интервал вправо увеличится более чем на 17 точек до края: расстояние между машинами поддерживается постоянным. Я решил сделать скорость машин растущей на 1 точку при каждой смене полосы: она равна 6 во второй полосе, 7 — в третьей...

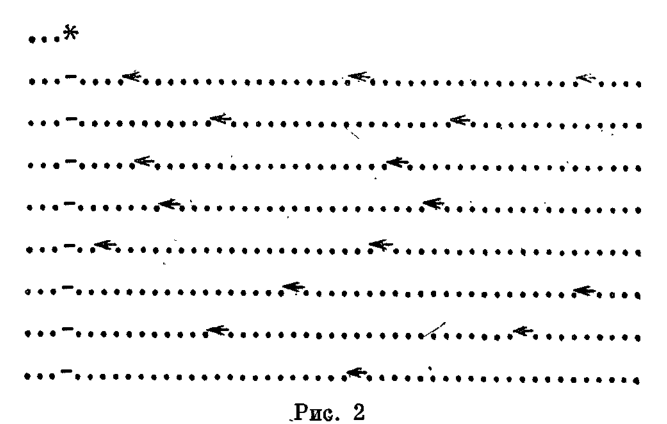


Рис. 2

Таким образом, расстояние между машинами растет, скорость тоже, но отношение не постоянно. Если вы вступаете на первую полосу в тот момент, когда машина только что проехала тире, то у вас 16 точек до машины справа от вас. При скорости 5 точек вы можете оставаться неподвижным три хода. На нижней полосе у вас осталось бы справа 23 точки, но при скорости 12 вы не можете оставаться на месте более одного хода. Чем дальше вы продвигаетесь, тем больше риск, что вы будете раздавлены.

При таком выборе данных период рисунка очень велик. У вас нет никакой возможности получить по ходу партии дважды одну и ту же конфигурацию.

Единственный случайный элемент: начальное положение машин на каждой полосе. Вы задаете это начальное положение, выбирая число точек между тире и первой машиной справа от тире; это — целое число, выбираемое случайно, строго меньше расстояния между машинами на данной полосе. Таким образом, для 8 полос нужно случайным образом получить 8 чисел. Используя воспро-

изводимую непредсказуемость последовательности, как это описано в разд. 1 вы можете переиграть партию, если сочтете, что плохо использовали ваши возможности. Вы можете провести соревнования со своими друзьями. В моей программе я подсчитываю число шагов, потребовавшихся для перехода дороги. Выигрывает тот, кто переходит с наименьшим числом шагов.

Не говорите: идиотская игра, придуманная в дурацком мозгу... Прежде всего это невежливо по отношению ко мне. Кроме того, в ней нужно иметь некоторый опыт, чтобы дать себе отчет в том, насколько трудно играть оптимальным образом. Благодаря изображению точек, вы можете — если захотите — проводить свои подсчеты и узнавать, каким будет положение машин после следующего хода. Вы можете предвидеть или вычислять столько ходов от начала, сколько вы пожелаете: игра вашего противника полностью определена. Но опыт показывает, что это скучно. Ходы лучше делать, оценивая положение машин перед следующим ходом. Может получиться, что вы говорите себе: у меня есть время пройти, а он возьмет да и раздавит. Может случиться, что вы не берете на себя риск пойти вперед, а машина останавливается в точности перед тире.

Точно так же может случиться, что вы правильно оцениваете ситуацию, но вам не удается достаточно точно рассчитать начальные ходы, и вы попадаете в ловушку. Вы оказываетесь на некоторой полосе и не раздавленным. Но нельзя ничего не делать: если оставаться на месте, то вас раздавят. Нельзя вернуться — там, на предыдущей полосе, машина слишком близко к тире. Нельзя идти и вперед: на следующей полосе машина стоит перед тире или слишком близко к тире.

Вот еще несколько предложений. Необходимо держать рисунок на экране неподвижным: только стрелки машин и крестик (×) пешехода должны перемещаться по неподвижному полю. Чтобы передвинуть пешехода, я предлагаю следовать очень простому правилу. На вопрос компьютера отвечать Н, если вы собираетесь пойти в нижнюю сторону (на само собой разумеющуюся полосу), В — если вы хотите перейти на полосу выше, и ничего не отвечать, если вы не хотите шевелиться.

Программирование этой игры очень просто. Желаю успеха.

* Игра 8. Шадок у гиби.

«У шадоков ситуация удовлетворительна. Испытания ракет продолжаются, постоянно кончаясь неудачами.

Дело здесь в одном из основных принципов шадокской логики: «Нет ничего, что бы непрерывно продолжалось и не кончилось успехом». Или, в других выражениях: «Чем больше неудач, тем больше шансов, что оно заработает». Их ракета еще несовершенна, но они вычислили, что у них есть по крайней мере один шанс из миллиона, что она заработает... И они торопятся поскорее осуществить 999999 первых неудачных опытов, чтобы быть уверенными, что миллионная заработает». (Жак Руксель. Великолепие навыворот. Париж, издательство Грассе.)

Великий колдун сказал, что ракеты терпят неудачу потому, что не хватает транзисторов в системах безопасности. Но у гиби транзисторы собирают с растений, произрастающих на огородах. Решено послать одного из шадоков на планету гиби искать транзисторы. Гиби, очень умные благодаря своим шляпам, быстро проникли в планы шадоков и решили позабавиться. Они позволили шадоку забраться в один из их огородов, но окружили его со всех сторон, и всякий раз, когда растение расцветает и дает транзистор, они мчатся, чтобы собрать урожай прежде шадока.

Вот вам тема игры. Как и в предыдущих играх, я предлагаю здесь версию, которую я реализовал на своем микрокомпьютере. Вы можете подогнать параметры в зависимости от возможностей вашей машины, а также в зависимости от желаемой трудности игры и шансов на успех. Было бы благоразумно. В первых версиях из осторожности стóит считывать все параметры до начала каждой партии. Вы сможете также сделать несколько попыток, чтобы добиться удовлетворительного расположения всех персонажей в огороде. Не придерживайтесь рабски сделанных ниже предложений.

В начале игры компьютер воспроизводит образ огорода, где появляются: гиби, обозначенные буквами Г; цветы с транзисторами, обозначенные цифрой, показывающий число транзисторов, которые можно собрать с этого цветка (есть цветки с одним транзистором, наименее продуктивные, и цветки с девятью транзисторами, наиболее продуктивные); шадок, представленный крестиком (×); пустые места, обозначенные точкой. Вот возможная комбинация. В ней 12 строк по 20 полей, с 15 гиби и 20 цветками. Все их значения указаны. Шадок имеет право на 40 ходов, чтобы собрать 100 транзисторов. Компьютер постоянно сообщает число оставшихся ходов и число уже собранных транзисторов.

На своем ходе шадок может переместиться на одну клетку в любом направлении. Я выбрал определение перемещения с помощью 0,1 и 4 букв: В — для верха, Н — для низа, Л — для левой и П — для правой стороны. Если ответ пуст, то шадок не шевелится. Если ответ П, то нужно сделать один шаг вправо на той же строке. Если ответ ВЛ (или ЛВ, порядок не важен), то шадок перемещается на 1 шаг по направлению диагонали вверх и влево.

Если при движении шадок оказывается на поле, занятом цифрой, то эта цифра исчезает из игры и ее значение прибавляется к сумме, набранной игроком. Случайным образом выбирается новая цифра и располагается на свободном игровом поле.

Ну, а теперь — о путешествиях гиби. Каждый гиби перемещается на один шаг по строке, столбцу или диагонали к ближайшей к нему цифре. Это правило может привести двух гиби на одно и то же поле. Есть много способов разрешить проблему этих столкновений: например, если поле назначения какого-либо гиби не является ни точкой, ни цифрой, то перемещение на него не осуществляется. Это проще всего. Если при своем перемещении гиби прибывает на поле, обозначенное цифрой, то эта цифра исчезает из игры. Когда все гиби перемещены, нужно случайным образом раздобыть столько же цифр, сколько было упразднено, и случайным образом расположить их на местах, обозначенных точками.

Игра кончается, либо когда шадок приобретает свои 100 транзисторов, либо когда число ходов, предоставленных ему, оказывается исчерпанным.

Эта игра включает намного более случайных элементов, чем предыдущая. Но можно играть с большей или меньшей ловкостью. На рис. 3 шадок может собрать урожай с одной из трех следующих цифр: с 3 — выше от него в том же столбце (этой цифре угрожает гиби, но шадок, играя первым, достигает ее раньше); с 5 — ниже и правее его (и ей тоже угрожает гиби, но шадок его опередит) с 9 — ниже и левее (эта цифра дальше, нужно три хода, чтобы достичь ее, вместо двух для предыдущих цифр, но нет ни одного гиби поблизости). Цифра 9 дает наибольшее число очков, но обходится в три хода. Если так и сделать, то уже ни одной цифры поблизости не окажется. Цифра 5 находится в углу, наводненном гиби. Как будто у нас нет особых оснований выбрать в качестве следующего хода что-либо другое. Именно 3 оказывается наилучшим выбором, потому что и 5, и 7, и 9 не слишком близко. В этом

углу гиби есть, но там и с цифрами неплохо, так что их перемещения более или менее предсказуемы (если сумеете, сделайте, чтобы это было в точности так). Итак, у вас есть возможность выбирать каждый отдельный ход наилучшим образом, но вы не можете проводить вычисления слишком

```
. . . . . . . 9 . . . . . . . . . . . .
. Г . 3 . . . . . 7 . . Г . . . . . . .
1 . . . . . . 5 . . . . . Г . Г . . 8 .
. . . x . . . . . . 4 . . . 3 . . . . .
. . . . . 5 . . Г . . . Г . . . 2 . . 1
. . . . . . . . . . 8 . . . Г . . . . .
. 9 . . . . . Г . . . . . . . . . 1 . Г
. . . . 5 . . . . . . . . 4 . . . . . .
. . . . . . . 7 . . . Г . . . Г . . Г .
. . . . . Г . . . . . . . . . . . . . .
. . 5 . . . . . . 2 . . . 8 . . . . . .
. . . . . . . Г . . . . . . . . Г . . .
```

Рис. 3

далеко: вы не знаете, как будут противодействовать гиби, а их достаточно много для того, чтобы при каждом ходе какие-то цветы исчезали, позволяя другим расцвести.

Эту игру не так уж трудно запрограммировать. Но нужно сосредоточить внимание на перемещениях гиби. Для каждого из них найдите ближайший цветок и, если несколько цветков находятся на одном и том же расстоянии, выберите случайным образом тот, к которому он отправится.

* И г р а 9. Плата за страх.

Шел когда-то фильм с таким названием. Я его не видел, но о нем достаточно много говорили по телевизору,

чтобы я знал, о чем он, и он дал мне идею гораздо менее опасной игры!

Вы — тот самый игрок, который, в обмен на обещанную кучу денег, рискует своей жизнью, которой угрожают наемные убийцы. Игра разыгрывается в пространстве, наполненном препятствиями. За вами гонятся трое

```
. . 0 0 0 . . 0 0 0 . 0 . . 0 . У 0 . .
0 . . . 0 . . . . . 0 . . . 0 . . 0 0 0
. 0 . . 0 0 . 0 . . 0 . . . . . 0 0 . 0
0 . . 0 . . . . 0 . 0 . . 0 0 . 0 . . 0
0 0 . 0 . . . . 0 . . . X . 0 0 0 0 0 .
. . . . . 0 0 . 0 . . 0 . 0 . 0 0 0 . .
0 . . . . . . . . . . 0 . 0 . . 0 0 . .
0 0 0 . . . . . . . . 0 0 0 0 . 0 0 . .
0 . 0 0 . . 0 . . У 0 . 0 0 . . 0 . . 0
. . . . 0 0 . . . 0 0 0 . . 0 0 . 0 . У
. 0 . . 0 0 . . . 0 . 0 0 0 0 0 . . 0 .
. . 0 0 0 0 0 . . . 0 . 0 . . 0 0 0 . 0
```

Рис. 4

убийц. Они вооружены револьверами и стреляют в вас, если вы с ними не разделены препятствием. Это — хорошие стрелки: если вы находитесь на линии выстрела, они не промахнутся и компьютер сообщит R. I. P. (requiescat in pace: «да покоится в мире» — для тех, кто совсем не учил латыни).

Более точно, игра снова реализуется на прямоугольнике, образованном точками (свободными местами) и нулями (препятствиями). Я выбрал прямоугольник с 12 строками и 20 столбцами. Я расположил там 100 препятствий и трех убийц (обозначенных У).

Рисунок 4 снят с экрана. Условимся, что убийцы могут стрелять только в направлении строки или столбца.

В приведенной конфигурации игрок (обозначенный ×) не находится на линии выстрела ни одного из убийц. Он может перемещаться на один ход в любом направлении (как король в шахматах). Игра разыгрывается следующим образом:

— игрок перемещается (один из способов перемещения — пребывание на месте, где он находится. Но можно помешать игроку укрываться в норе. Я ограничиваю число стояний на месте пятью ходами). Переходить можно только на место, обозначенное точкой. Если игрок оказывается после этого на линии выстрела одного из убийц (в той же строке или в том же столбце и не огороженным препятствием), то он мертв;

— после этого трое убийц перемещаются на один шаг — все равно в каком направлении (они не могут оставаться неподвижными). Они перемещаются на поле, обозначенное точкой. Нужно договориться об их перемещении, чтобы учесть в случае необходимости спорные ситуации, например, перемещать их одного за другим, что позволяет для каждого из них учесть движения предыдущих. Убийцы, когда у них есть возможность, перемещаются так, чтобы приблизиться к игроку. Если в результате этого перемещения убийца оказывается в состоянии взять игрока на мушку, то он стреляет и убивает его. Игра сразу кончается. Если это не так, то цикл возобновляется.

Если игроку удается просуществовать в продолжение данного числа ходов, он выигрывает.

Может случиться, что убийца оказывается бок о бок с игроком, но по диагонали. Он не может стрелять, потому что не находится ни на той же строке, ни в том же столбце. Вы можете сказать, например, что ваш игрок — чемпион по дзюдо и что убийцы не рискуют атаковать его в ближнем бою. Но вы можете принять и противоположную тактику: если при разрешенном перемещении убийца может попасть на клетку игрока, то последний считается убитым. Тем самым вы уменьшите шансы игрока...

Эту игру запрограммировать не очень трудно. Нужно только принять единственную меру предосторожности: в процессе бросания жребия о начальной конфигурации устройте так, чтобы ситуация не оказалась катастрофической с самого начала игры: ни один из убийц не должен находиться ни в строке, ни в столбце, где находится игрок, а также и не в соседних строках и столбцах.

Я сыграл немало партий. Есть два способа играть,

Можно трепыхаться в набитом препятствиями участке и плавать между двумя соседними неприступными полями. Выигрываешь без славы... Можно обыгрывать трудности и, напротив, пытаться вовлечь убийц в гонку преследования, уклоняясь от всех их ловушек. Это намного труднее. Их все-таки трое... Если у вас появятся соображения о том, как ограничить возможности избирать первую тактику, используйте их. Я в этом не преуспел. Деятельность по подсчету стояний на одном месте — это простейшая защита, позволяющая избежать случая, изображенного на рис. 5. Попав однажды на место, обозначенное крестиком (×), игрок может оставаться там бесконечно. Перед лицом необходимости перемещаться убийцы то освобождают, то снова занимают два места, обозначенные буквой У, но не имеют возможности выселить игрока. Если же число стояний на месте ограничено, то игроку невыгодно входить на это поле, с которого он больше не сможет уйти. Но это может оказаться выгодным в конце партии, если число оставшихся ходов меньше числа разрешенных стояний на месте.

```
. У О У .
О О х О .
. О О О О
```

Рис. 5

Игра 10. Игра роботов.

Я принял за образец игру, которую я нашел в обзоре по компьютерам. Я глубоко сожалею, что не узнал о ней больше, чтобы воздать ее автору (мне неизвестному) по заслугам. Ее тема в каком-то смысле сравнима с темой «платы за страх», но правила другие и они дают существенно отличающуюся стратегию игры при не очень измененном программировании. Это — идеал для тех, кто больше любит играть с компьютером, чем писать программы. Здесь мы отличаемся: для меня большее развлечение — писать программы...

История происходит в 2387 году. Космическая экспедиция достигает планеты X. Один из участников экспедиции проникает в огромный зал разрушенного здания. Земля изрыта многочисленными расщелинами, открывающими бездонные пропасти. Ни одной живой души, но местные жители достигли высокого технического уровня. Они построили автоматические заводы, производящие движущихся роботов. Заводы еще работают сами по себе, но с перебоями. Появление роботов случайно, да и не работают больше эти роботы так, как когда-то... Они продолжают стремительно нападать на пришельцев, но как слепые. Если избранный ими путь приводит их к расщелине, они

оказываются не в состоянии избежать ее и проваливаются в дыру. Что же касается избираемого ими пути, то он определен полностью без всяких уловок: прямо к пришельцу. Игра начинается с входа посетителя в помещение. Дверь за ним автоматически закрывается. Единственный

```
. . . . 0 0 . . 0 . P . . P 0 . P .
. 0 . . . . . . . . . . . . . P . .
. P P 0 . . . . . . . 0 . . . . . .
. . 0 . . . 0 . . . . . . 0 P . . P
. . . . . . . . 0 . . . P P . . . 0
+ . . . 0 . . . . . . P . 0 P 0 . X
. . P 0 0 . . . . . P 0 . . . . . .
. . P . 0 . . . . . 0 . . . . P . .
. . . . . P 0 . . . . . 0 . 0 P . .
. . . . 0 . . 0 . . . . . 0 . . 0 .
. . . . P . . 0 0 . . . . . . 0 . P
```

Рис. 6

выход — на другом краю. Роботы входят в зал через четыре угла. В начале игры в помещении находится некоторое количество роботов. У посетителя есть два козыря:

— с помощью хитроумных перемещений он может заставить роботов сваливаться в расщелины и тем самым отделываться от них;

— у него есть несколько дезинтегрирующих зарядов, с помощью которых он может разрушать роботов. Но он может применять их только в ближнем бою (разве что дальность его оружия не ограничена. Насколько мы знаем, на земле такая дистанция есть...). Кроме того, ему нужно экономить заряды. Еще неизвестно, что его ждет, когда он приблизится к выходу...

Рисунок 6 воспроизводит экран микрокомпьютера. Помещение есть прямоугольник с 11 строками и 18 столбца-

ми. Строки обозначают свободные места, 0 — расщелины в полу, Р — роботы. Крестик (×) обозначает игрока, здесь — в начальном положении. Выход обозначен плюсом.

При своем ходе игрок может

— убить роботов на полях, прилегающих к его собственному;

— переместиться на одно поле в любом направлении, при условии, что он не попадет в расщелину, в результате чего он погиб бы. Он не должен также перемещаться в клетку, помеченную Р, так как там он был бы уничтожен роботом. Если это перемещение приводит его к полю +, то он выигрывает.

Вот что сделал я для реализации этого. Игрок сначала говорит, каких роботов он хочет разрушить, а затем — куда он хочет пойти. Если он отвечает

—Л, —В, НЛ

это означает, что должен быть разрушен робот слева (на соседнем поле), как и робот, находящийся на соседнем поле над ним, после чего он передвинется на поле, расположенное ниже и левее.

Каждый раз, прочитав ответ, компьютер вычисляет новое состояние игры и показывает его. Число роботов, которых игрок может разрушить, ограничено, компьютер следит за ним и в каждое мгновение может сообщить.

После этого роботы переместятся, каждый на одну клетку, все равно в какую сторону (по горизонтали, по вертикали или по диагонали). Если при этом робот оказывается на поле 0, то он уничтожается. Если два робота сталкиваются при движении, то один из них уничтожается. Кроме того, роботы случайным образом добавляются в четырех углах игрового поля. Это нужно делать для того, чтобы число роботов в игре оставалось более или менее постоянным.

В случае, изображенном на рис. 6, игрок может остаться неподвижным. Тогда робот, расположенный над игроком в том же столбце, уничтожается в западне, расположенной непосредственно ниже, но робот, расположенный левее и ниже игрока, приблизится к игроку, и последнему придется разрушить его на следующем ходе. Игрок может также начать с хода влево, но тогда два робота приблизятся к нему, и ему придется разрушить их на следующем ходе.

Игра оказывается более или менее трудной в зависимости от соотношения между числом препятствий и числом роботов. Я взял прямоугольник с 11 строками и 18 столбцами (число строк нечетно по причине особой роли, которую играет среднее поле), с 30 расщелинами и 20 роботами. Игрок имеет право разрушить 12 роботов. В начале игры игроку, как правило, трудно выйти из своего начального положения, потому что он находится поблизости от двух правых углов, из которых появляется немало свежих роботов. Когда же ему удается удалиться от правого края, выходящие из углов роботы его меньше стесняют и большая их часть падает в расщелины. Трудности возобновляются при приближении к левому краю. Именно поэтому нужно избегать растраты боеприпасов в начале партии. Попытайтесь, и вы увидите, что это требует немалой ловкости...

* И г р а 11. Формула 1 *).

Задумывались ли вы когда-нибудь над тем, что переживает водитель, мчащийся на огромной скорости по извилистой дороге, обгоняя попутные машины и уклоняясь от встречных? Конечно, вы не преобразуете ваш компьютер в быстро мчащийся автомобиль, и с помощью используемых нами графических средств мы не сможем создать впечатление движения по расстилающейся перед вами дороге. Так как, наконец, я полагаю, что у вас нет средств управления вашим компьютером в реальном времени (этих средств нет не только у меня, но и в оборудовании учебных заведений) **), поэтому перед любым вашим действием вы сможете размышлять столько времени, сколько вам захочется, а это совсем не так в случае водителя на дороге. Но попытайтесь-ка в этой игре реагировать быстро и вы увидите, что эффект не так уж плох, несмотря на элементарность средств...

Автострада выводится на экран в виде последовательности строк, на которых поставлены точки, нули и звездочка. Точки реализуют 4 полосы движения и представляют свободные места. Выше транспортное средство представимо звездочкой. Нули суть неподвижные препятствия (скажем, что-то тяжеловесное и очень медлительное). Типичная ситуация изображена на рис. 7. Вы находитесь перед участком дороги (я выбрал 13 строк. Это дает

---

*) Так начинаются правила проведения автогонок.— *Примеч. ред.*

**) Напомним, что книга написана в начале 80-х годов.— *Примеч. ред.*

мне охрошие результаты. Но вы можете взять больше, если вам позволяет ваш экран; это увеличивает возможности предвидения. Вы можете взять и меньше, что заставит вести машину в еще более стесненных условиях...).

У вас есть некоторая скорость, которая не выводится на экран, но вы можете ее узнать. В начале игры решается,

```
  . * . .
  . . . .
  . . . .
  . . . .
  . . . .
  . . . .
. . . 0
. . . .
  . 0 . .
  . . . .
  . . . .
    . . . .
      . . . .
```

Рис. 7

```
    . * . .
  . . . 0
  . . . .
    . 0 . .
    . . . .
    . . . .
      . . . .
        . . . .
      . . . .
    . 0 . .
  . . . .
. . . .
  . . . .
```

Рис. 8

фиксируется ли она сразу на всю игру (я выбрал 4) или предлагается вами. Когда вы решаете — ускоряете ли вы движение или замедляете его — вы можете узнать на каждом ходе, какова она. Но я не считаю уместным заставить компьютер сообщать ее постоянно — это слишком многое облегчает водителю. Увидите сами.

Компьютер требует от вас, что вы собираетесь делать: ускорять, замедлять, повернуть правее или повернуть левее. Попарно эти возможности взаимоисключающие, но вы можете одновременно ускорить движение и взять вправо или замедлить и взять влево. Вы можете также

не менять ни вашу скорость, ни направление вашего движения.

Если вы ускоряете движение, то ваша скорость увеличивается на одну единицу. Если вы замедляете движение, то скорость уменьшается на единицу. Если вы таким образом добираетесь до нуля, то вы рассматриваетесь как проигравший партию: вы не имеете права останавливаться...

Предположим сначала, что вы не меняете направления движения. Ваша машина спускается по вертикали на число строк, равное вашей скорости. Это может привести к тому, что вы пересечете поле, обозначенное 0. Тогда вы сталкиваетесь с грузовиком, вы пропали. Это может также привести вас к тому, что вы попадаете на не обозначенное поле. Вы покинули дорогу. Это — тяжелый несчастный случай. Вы пропали. Так, на рис. 8, если ваша скорость превосходит 2, то вы сталкиваетесь с грузовиком. Если бы вы исходили из крайнего левого поля того же ряда и ваша скорость превосходила бы 5, то вы покинули бы дорогу (внимание: вы пропали с того момента, как вы достигли нуля или не обозначенного поля. Оставшаяся часть вашей траектории движения не рассматривается).

Теперь, если вы меняете направление движения, двигаясь, например, вправо, то ваша машина продвигается по диагонали из исходного ряда вправо до следующей строки, а затем движение продолжается дальше в том столбце, в котором оказывается машина. Понятие «вправо» двусмысленно: вы можете расширить его до «вправо на фигуре» или «вправо по направлению движения». Это не так уж важно, выберите тот смысл, который вы пожелаете. Если это вас шокирует, переверните рисунок и заставьте машину подниматься; тогда «направо» будет значить «направо на экране» во всех случаях. Но это немного усложняет программу, и я так не делаю.

Если вы повернете направо в случае, изображенном на рис. 8, то какова бы ни была ваша скорость, вы сразу же сталкиваетесь с грузовиком. Если вы повернете налево, то вы избегаете грузовиков, но ваша скорость не должна превосходить пяти.

Когда вы сообщили все ваши команды, компьютер показывает положение звездочки на последовательных строчках, чтобы материализовать ваше движение. Если вы оказываетесь на поле, не помеченном точкой, то все останавливается, вы пропали. В противном случае через

некоторое время — время ожидания, дающее вам возможность лучше рассмотреть пройденное вами, вся фигура поднимается, чтобы возвратить вашу машину на верхнюю строку, и на экране появляется новый кусок дороги. Таким образом, вы можете обнаружить, что вы едете слишком быстро и что дорога резко поворачивает или что попарно гуськом идут по два тяжелых грузовика, которые блокируют две полосы движения. Вы можете затормозить, но только на одну единицу. Придется рисковать...

Для того чтобы игра не продолжалась бесконечно, вы должны условиться о числе линий, которые нужно пересечь (например 100). Если вы достигли этого, компьютер прославляет вас так, как вы того заслуживаете, и указывает вашу среднюю скорость.

Эту игру программировать не очень трудно, если не считать того, что нужно оказаться способным корректно дозировать число грузовиков, и что нельзя позволять дороге часто делать зигзаги, а также выходить за пределы экрана. Но здесь нет ничего, с чем вы не могли бы справиться.

Собственно игра оказывается гораздо труднее, чем можно было себе представить. Вы, конечно, можете затормозить до скорости 1. Вы оказались на краю. Но ничего смешного нет. Вы можете оказаться в самом рискованном положении, вы терпите удар за ударом. Благодаря воспроизводимым непредсказуемым последовательностям вы можете много раз возобновлять один и тот же пробег. Впрочем, здесь бывает трудно вспомнить, что же происходит на шестидесятом километре... Вы можете также попробовать маршрут, а затем предложить его вашим друзьям. И если вы на их глазах вылетели с трассы, но вовсе не факт, что они смогут на ней удержаться. Итак, желаю успеха!

?** Игра 12. Твоя песенка спета, любопытный!

Идея не нова, да и реализация поступила в рыночную продажу. Но в этой игре возникают некоторые маленькие задачи по программированию, и я предлагаю вам красивый план их реализации. К тому же это позволит ввести вас в игры с числами.

Вы знаете телевизионную игру: вытащить случайным образом 6 шашек среди 24, образованных следующим образом:

двойной набор из 10 шашек с числами от 1 до 10;
четыре шашки с числами 25, 50, 75, 100.

Случайным образом выбирается трехзначное целое число (первая цифра которого — не нуль, так что оно содержится между 100 и 999, включая границы). Задача состоит в том, чтобы менее чем за 45 с обнаружить последовательность операций, использующих только значения шести выбранных шашек, причем каждую не более одного раза, и соединить их знаками + — × / (целочисленное деление разрешается только в тех случаях, когда оно выполняется нацело, без остатка).

Вот пример, который я получил с помощью своей программы.

Шашки: 4 4 7 8 9 100

Число, которое нужно получить: 380

В течение 45 с, которые я выделил своему компьютеру, я получил следующее решение:

$4 \times 100 = 400$
$9 + 8 = 17$
$7 + 17 = 24$
$24 - 4 = 20$
$400 - 20 = 380$

Это решение использует 6 шашек. Компьютер сообщает еще через 45 с:

$4 \times 9 = 36$
$4 + 36 = 40$
$7 \times 40 = 280$
$280 + 100 = 380$

Это решение не использует шашки 8.

Не пытайтесь сделать эту программу, следуя методу игры: вытащить случайным образом 6 шашек, вытащить случайным образом число, которое нужно получить, сообщить и то, и другое и в продолжение следующих 45 с искать нужную комбинацию. У вас нет никаких шансов, чтобы это произошло (см. Головоломку 28). Действуйте лучше следующим образом.

Выберите случайным образом 6 шашек и их комбинацию. Если результат не лежит в промежутке от 100 до 999,— повторите выбор. Если результат допустим, то выведите сообщение, какие 6 шашек участвуют, расположив их, например, в возрастающем порядке, чтобы не было понятно, в каком порядке они были использованы; сообщите искомое число, затем сообщите оставшиеся секунды и, когда 45 с протекут, сообщите результат. Здесь

есть неудобство: всегда есть хотя бы одно точное решение. И при том, что не приходится особенно обольщаться, то, что вы достигнете с его помощью, вы, может быть, сможете сделать по-другому только приближенно.

Внимание: случайным образом выбирать комбинацию на самом деле вовсе не всегда так просто, как в приведенном примере. Не забывайте, что вы можете использовать и не все шашки. Найдите способ получать ответ. Я не

```
    . . .
    . d *
a * . . e . g
. b c . . f .
h . j k l m n
    o p q
    r s t
```

Рис. 9

```
    . . .
    . . .
. . * . * . .
a b c d e f g
h i j k l m n
    o p q
    r s t
```

Рис. 10

вполне удовлетворен своим собственным. Я предпочел бы знать и другие способы это сделать...

?** Игра 13. Две лисы и 20 кур.

Когда я был молод, мы играли в эту игру на свежем воздухе, используя маленькие булыжники в качестве кур и два булыжника побольше для лис. Мы расчерчивали эту игру мелом на асфальте или палкой на утрамбованной земле.

Вот как я представляю эту игру на экране своего компьютера. Буквы представляют кур, звездочки — две лисы. Куры могут перемещаться на один шаг вверх, влево или вправо, но не назад и не по диагонали. Лисы также могут перемещаться только на один шаг, но также и вверх — как и вниз, влево и вправо. Лиса может съесть курицу — как в игре в шашки: если в горизонтальном или вертикальном направлении за курицей на один шаг следует свободное поле, то лиса перепрыгивает через курицу на свободное поле и берет ее. При этом трофеи складываются. На рис. 9 одна лиса может съесть курицу b, тогда как вторая лиса может съесть за один ход кур e и f. Лисы всегда обязаны есть и, когда у них есть выбор — как на рис. 9,— они обязаны осуществить наиболее длинное поедание. Если два приема пищи имеют одинаковую длину, осуществляется один из них — по выбору лисы.

В запрограммированной версии компьютер играет за лис. Вы перемещаете кур. Партнеры играют по очереди, причем куры начинают. Они выигрывают партию, если девяти из них удается занять 9 полей, образующих верхний квадрат игры (квадрат, нижние углы которого на рис. 10 занимают лисы). Начальное положение кур и лис изображено на рис. 10. Куры выигрывают также, если им удается заблокировать лис.

Лисы выигрывают, если им удается съесть 12 кур, так как тогда оставшихся кур недостаточно, чтобы занять 9 верхних полей.

Может показаться удивительным, что я отношу эту игру к категории игр без стратегии. Как вы собираетесь перемещать лис по вашей программе для компьютера? Действительно, возможностей слишком мало, и едва ли стоит говорить о стратегии. Нужно, чтобы при каждом ходе программа искала наиболее длинный среди всех возможных путь поедания для лис и осуществляла его, если он единствен. Если существуют два таких пути, то один из них нужно выбрать Если их нет совсем, то способ действия состоит в том, чтобы посмотреть, позволит ли какое-нибудь перемещение лисы поставить ее в состояние возможного поедания. Если такой ход есть, то почему бы его не сделать, это заставит кур реагировать. Если и такого угрожающего хода нет, то остается мало возможностей выбора. Я был поражен, увидев, что если выбрасывать ходы случайным образом вместо того, чтобы осуществлять их выбор, то результат будет не намного хуже... Но, конечно, не так уж трудно придумать что-нибудь получше. Единственная настоящая трудность программирования — определение наиболее длинного пути поедания.

?** Игра 14. Одна лиса и 13 кур.

Это — вариант предыдущей игры. Та же конфигурация, но только одна лиса и 13 кур. Та же задача: 9 кур должны занять верхний квадрат. Лиса обязана есть, и притом по наиболее длинному пути.

В отличие от предыдущей игры лиса и куры могут также перемещаться по диагонали, но куры не могут двигаться вниз. Линии на рис. 11 указывают на возможные перемещения.

Программирование этой игры сравнимо с программированием предыдущей. Я попытался быть немного более хитрым при определении перемещений лисы: так как здесь меньше того, за чем нужно следить, то можно из-

росходовать немного больше времени, чтобы заняться единственной лисой.

В результате получилась более хитрая игра. В варианте с двумя лисами курам довольно легко удается блокировать лис и, таким образом, выиграть. В версии с одной-единственной лисой увеличивается богатство возможных перемещений, и блокировать лису трудно. Можно отдать не более четырех кур и не так-то легко пожертвовать их так, чтобы отправить лису на другой край, в то время как остальные куры заполняют курятник.

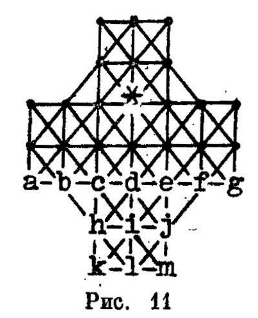


Рис. 11

Привыкнув к игре с двумя лисами, я вначале никак не мог приспособиться к этой игре, особенно к манере лисы ходить по диагонали. Но это не страшно. С того момента, как вы полностью ухватите способ движения (и, в частности, возможность перейти из *h* в *b* и из *j* в *f* на рис. 11), эта программа даст вам настоящую возможность играть: вы можете применить ту или иную стратегию игры на выигрыш против машины, которая такой стратегией не очень-то обладает...

Игра 15. Игра Доминика.

Уж здесь-то я могу ручаться, что это игра для начинающих. Ее для своего малюсенького микрокомпьютера придумал мой племянник Доминик. Она напоминает «плату за страх» (игра 9). Начальное положение игры — то же (рис. 4). Доминик взял прямоугольник поменьше и уменьшил число препятствий. Для начала он поставил препятствия на определенные места.

Правила игры изменены. Убийцы не вооружены огнестрельным оружием, у них — только ножи. Они не могут добраться до игрока иначе, чем достигнув занимаемого им поля. Игрок перемещается на 1 шаг в любом направлении (по горизонтали, по вертикали, по диагонали) с условием перемещаться на свободное поле. Убийцы на своем ходе приближаются к игроку на один шаг — обязательно на свободное поле — в любом направлении.

Игра осталась очень интересной. Проблема нахождения игрока и убийцы на смежных линиях больше не стоит. Если убийца оказывается рядом с игроком в каком-либо направлении, он его хватает на следующем ходе, если игрок не удаляется от него при своем ходе...

4. ИГРЫ СО СТРАТЕГИЕЙ

В этом разделе мы предлагаем программировать игры, главная трудность которых заключается в том, чтобы дать компьютеру хорошую стратегию. Разделение на игры со стратегией и без нее до некоторой степени произвольно. Уже по поводу случайных чисел мы предлагали игру со стратегией (игра 2). Конечно, совершенно необходимо, чтобы вы могли хоть немного развлечься... Некоторые из игр, с которыми вы познакомитесь, требуют не намного больше размышлений, чем игра в лис и кур. На самом деле это во многом зависит от особенностей вашего ума: стратегия, очевидная для одного, является головоломкой для другого.

Можно также упрекнуть некоторые игры в том, что они теряют всякую привлекательность, поскольку компьютер располагает выигрывающей стратегией. Если партнер компьютера не знает этой стратегии, то он проигрывает все партии. Если же он ее знает, то тот, кто делает выигрывающий начальный ход, неминуемо становится победителем.

Наконец, некоторые игры настолько распространены, что я постеснялся их здесь предлагать: такова, например, игра Отелло. Не пытайтесь браться за шашки или шахматы. Слишком трудно!

В этом разделе вам предстоит встретиться с двумя трудностями:

— найти стратегию для компьютера;

— запрограммировать ее.

Именно потому, что есть так много опубликованных плохих программ, я, не ссылаясь на эти публикации, добавил игру Нима. Сочувствую всем тем, кто может оказаться обиженным. Но тот, кто публикует программы, всегда рискует. Таковы правила этой игры.

Игра 16. Чтобы войти в курс дела.

Это — крайне простая игра, которую Роуз — Бол [BAL] относит к средневековым играм, поскольку ей действительно более 500 лет.

Вы выкладываете на стол 50 спичек. Каждый игрок по очереди вынимает спички из кучи, по меньшей мере 1 и не более 6. Кто берет последнюю спичку, выигрывает.

Вы можете реализовать ее, заставляя компьютер сообщать число оставшихся спичек. Когда очередь хода за компьютером, он делает ход настолько быстро, что игрок не успевает увидеть происходящего. Включите в вашу

программу «цикл ожидания», чтобы замедлить игру компьютера (цикл от 1 до нескольких тысяч, в котором ничего не происходит:

ДЛЯ $i = 1$ ДО 2000 ВЫПОЛНЯТЬ ВЕРНУТЬСЯ)

Вы можете изменить игру, взяв в качестве допускающих изменение данных — начальное число спичек и максимальное число спичек, которое можно вытащить на каждом ходе.

? Игра 17. Игра дат.

Эта игра предложена Берлокеном [BER]. Номер года в ней не очень существен, но предполагается, что год не високосный: в феврале 28 дней. Первый игрок сообщает какую-нибудь дату января. Каждый игрок на своем ходе называет более позднюю дату, увеличивая либо календарную дату в месяце, либо месяц, но не то и другое сразу. Если, например, начальной датой было 8 января, то можно перейти к 8 марта или к 12 января. Можно увеличить меньше: 9 января или 8 февраля; можно перейти сразу к 8 декабря или 31 января. Внимание: если вы переходите к 31 января, то ваш противник сможет в дальнейшем менять только месяцы, и притом лишь месяцы с 31 днем.

Первый, кто доберется до 31 декабря, выигрывает.

У вас не должно возникнуть никаких затруднений ни в определении стратегии, ни в программировании этой игры. Подумайте о проверке осмысленности предлагаемых дат... Кроме того, вставьте цикл ожидания, чтобы дать игроку время для ответных действий. Компьютер должен быть вежлив и должен спросить, кто будет начинать, по крайней мере, бросить «орла» или «решку», чтобы узнать, кому начинать...

?** Игра 18. Игра с 24 картами.

Расположим на столе 24 раскрытые карты: все карты с номерами от 1 до 6 обычной колоды, где туз считается за 1. Масти карт несущественны: двойка бубен имеет то же значение, что и двойка треф.

Каждый игрок при своем ходе берет со стола карту и складывает ее значение с суммой тех, которые были взяты ранее *). Первый, кто берет в точности 50 очков, выигрывает. Внимание: если при вашем ходе вы, взяв карту, не можете не превысить 50 очков, то вы проиграли. Если,

---

*) Таким образом, подсчитывается общая сумма карт, взятых партнерами, а не отдельные суммы для каждого партнера.— *Примеч. ред.*

например, ваш партнер увеличил сумму до 49 очков, а все тузы уже взяты, то вы проиграли: карту нужно брать, а ее значение больше единицы.

Это — вариант средневековой игры. Стратегия гораздо сложнее, потому что карт каждого сорта только 4. К этой игре нужно привыкнуть. Сперва компьютер выигрывает все партии подряд (любопытна реакция программиста: я счастлив, что моя программа меня обыгрывает). Но по прошествии нескольких партий уже я выигрываю. Тогда программу нужно улучшить.

?** Игра 19. Игра города Нима.

Ах! Эта нимская игра... кто ее на знает. Существует немало запрограммированных вариантов в большом числе публикаций и обзоров. Читатель может сказать мне, что этой игре здесь не место: если моя цель — заставить читателя программировать, то я проиграл с самого начала.

Ну, нет. Поскольку решения, предложенные в вышеупомянутых книгах (и, поскольку перечитывать их бесполезно, то я их имена и не указываю), очень плохо запрограммированы и совершенно не объяснены. Вам придется сделать лучше. Если вы знаете выигрывающую стратегию, то вам придется ее испытать, чтобы её проверить. Если вы ее не знаете, вы должны попробовать ее изобрести. Во всех случаях нужно программировать очень тщательно, чтобы не делать ненужных вычислений.

Напомним даже саму игру на тот очень мало правдоподобный случай, если вы ее еще не знаете. Это игра для двоих, и компьютер будет вашим партнером. На столе — кучи спичек в некотором количестве, и в каждой куче — некоторое количество спичек. Например, есть 5 кучек с 8, 13, 7, 5, 9 спичками.

Каждый игрок на своем ходе берет столько спичек, сколько хочет, из одной кучки, но он обязан взять хотя бы одну. Выигрывает тот, кто берет последнюю спичку. Вот партия, сыгранная от начала до конца. Компьютер начинает.

Исходное положение: 8, 13, 7, 5, 9

Ход компьютера	Ваш ход
6, 13, 7, 5, 9	6, 3, 7, 5, 9
6, 3, 7, 5, 7	2, 3, 7, 5, 7
2, 3, 7, 1, 7	2, 0, 7, 1, 7
2, 0, 4, 1, 7	2, 0, 3, 1, 7
2, 0, 3, 1, 0	2, 0, 2, 1, 0
2, 0, 2, 0, 0	

Вы проиграли. Если вы возьмете две спички из одной кучки, то компьютер возьмет две из другой, так что и последнюю и потому выиграет. Если же вы возьмете одну спичку иэ одной кучки, то он возьмет одну из другой, и вы проигрываете на следующем ходе.

Мариенбадская игра является простым вариантом этой: проигрывает тот, кто берет последнюю спичку...

Этот род игр можно решительно осудить. Это — совершенно несправедливая игра. Выигрывающая стратегия

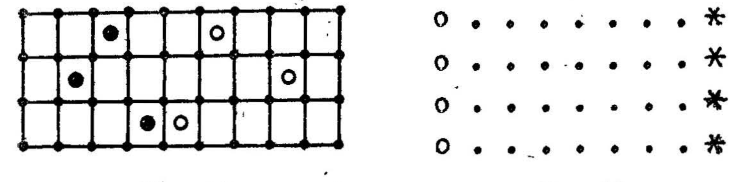


Рис. 12

Рис. 13

существует. Если ваш противник ее не знает (как было в приведенном выше примере), то он обязательно проигрывает. Если же он её знает, то он первый воспользуется усвоенной им выигрывающей стратегией, и вы ничего не сможете сделать. С другой стороны, даже если вы знаете выигрывающую стратегию, вы рискуете проиграть компьютеру, потому что вы не так хорошо считаете...

?** Игра 20. Игра Норткотта.

Вот менее известная игра, которую, однако, гораздо труднее программировать. Эта игра разыгрывается двумя участниками на прямоугольной площадке, разделенной на поля, как показано на рис. 12.

Каждый игрок располагает по шашке на каждой строке. В начале черные шашки находятся на левых полях, белые — на правых. При каждом ходе игрок перемещает одну из своих шашек направо или налево на столько полей, сколько он хочет; но он не может переходить край игрового поля, и не может переходить за клетку, предшествующую противоположной шашке; шашки друг друга не берут и нельзя переходить занятое поле. Проигрывает тот, кто не может пошевелиться, потому что все его шашки загнаны между краем и противоположными шашками.

Размер игры значит очень мало. Я предпочитаю игру с тремя строками, но 4 и 5 — тоже очень хорошие числа. Длина строк несущественна. Выберите ее так, чтобы игра хорошо смотрелась.

На экране вы можете воспользоваться техникой, которая нам так часто служила. Пусть свободные клетки будут представлены точками, шашки одного из игроков — звездочками, а другого — 0. На рис. 13 воспроизведено начальное положение (4 строки, 9 полей на строке). Ход компьютера состоит просто в перемещении одной из его шашек (внимание: если ответ будет слишком быстрым, его, может статься, будет трудно воспринимать. Подумайте, как использовать цикл ожидания). Ход игрока может быть дан компьютеру в виде указания, на какой строке нужно переставить шашку и число полей при перемещении; положительное число указывает на приближение к противнику, отрицательное число означает отход к краю игрового поля. Все это очень просто.

?* Игра 21. Игра Кейлеса.

В наиболее простой форме эта игра разыгрывается со спичками, положенными в один ряд. Каждый игрок на своем ходе вынимает либо какую-то одну спичку из строки, либо две смежные спички. Это может разломать исходный ряд на несколько меньших рядов. Вот, например, начальная конфигурация (спички обозначены нулями), а затем — состояние игры через несколько ходов (точки обозначают места, оставшиеся пустыми).

Вначале:

0 0 0 0 0 0 0 0 0 0 0 0 0 0

После нескольких ходов:

. 0 0 0 . . 0 0 . 0 0 0 0 . .

Выигрывает тот, кто берет последнюю спичку.

Игру можно легко распространить на случай нескольких исходных линий спичек. На каждом ходе игрок берет либо одну спичку, либо две соседние спички на линии, которую он выбрал.

Как и в предыдущих играх, подумайте о применении цикла ожидания, чтобы у вас было время увидеть ответ компьютера. Как и в играх Нима и Норткотта, эта игра не очень-то справедлива. Компьютер выигрывает все партии, по крайней мере, если его противник не знает выигрывающей стратегии...

* Игра 22. Игра Сима.

Еще одна игра, тоже совершенно не равноправная для двух игроков — для компьютера и для вас.

Игра разыгрывается на листе бумаги, на котором обозначены 6 точек, являющихся вершинами правильного

шестиугольника, помеченные буквами A, B, C, D, E, F. Естественно ожидать, что вашим партнером будет компьютер, и ему никакой бумаги не нужно, так что 6 точек появятся на экране.

Каждый игрок при своем ходе проводит отрезок прямой, соединяющий еще не соединенные вершины. Нужно, чтобы следы отрезков, проведенных различными игроками, можно было отличить. На рис. 14 следы одного — сплошные, а у другого — штриховые. В запрограммированной игре именно компьютер берет на себя проведение отрезков. Скажите ему только названия вершин, которые вы хотите соединить. Он и проведет отрезок, причем такого типа, который вам присвоен. Затем он выберет вершины, которые он захочет соединить, и проведет свой собственный отрезок.

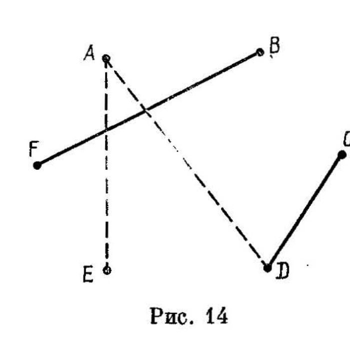


Рис. 14

Проигрывает тот, кто первым построит треугольник из своих собственных отрезков. Так, на рис. 14 тот, кто проведет отрезок из тире между D и E, проигрывает, потому что он образует отрезок AED из тире, между тем как отрезок AC из тире безопасен: он образует, конечно, треугольник ACD, но одна из его сторон — сплошная, и поэтому он безвреден для обоих игроков.

Сумеете ли вы показать, что в этой игре всегда есть проигравший? (Нельзя так расположить все возможные отрезки, чтобы никакие три стороны их, одинаковым образом выполненные, не образовывали бы треугольника.) Сумеете ли вы предложить хорошую стратегию для компьютера? Если компьютер и игрок играют в равную силу и не совершают никаких ошибок, кто тогда выиграет? Начинающий? Или другой?

На моем микрокомпьютере, не имеющем графических средств, я был вынужден удовлетвориться проведением отрезков с помощью выстраивания в ряд букв там, где компьютер считал нужным их поставить. Картина получалась не очень красивой, но игра оставалась осуществимой и интересной. С графическим да еще с цветным экраном у вас должно получиться просто загляденье. Но не забывайте поговорку Анри Ледгара [LED]:

Не занимайтесь формой вывода результатов, пока ваша программа не окажется правильной.

Когда ваша программа окажется правильной, тщательно отработайте форму вывода результатов.

? Игра 23. Спички Бергсона.

Нет, этот великий философ не играл в такие игры — по крайней мере, насколько я знаю. Эту игру предложил мне М. Дюма, профессор лицея Бергсона. Еще одна игра для двоих, в которой компьютер — ваш неумолимый партнер. Потому что если вы обнаружите выигрывающую стратегию, то компьютер не оставит вам никаких шансов, ведь у него и память есть...

На столе — кучка спичек (достаточно большая: вначале — по крайней мере 50). Каждый игрок при своем ходе берет спички из кучки. Нужно взять по крайней мере одну и не более чем вдвое больше, чем взял предыдущий игрок. На первом ходе можно взять одну или две спички. Выигрывает тот, кто берет последнюю спичку.

Вот последовательность возможных ходов. Вначале — 50 спичек.

Игрок А берет	Остается	Игрок Б берет	Остается
1	49	2	47
4	43	8	35
16	19	19	выиграл

На каждом ходе, кроме последнего, каждый из игроков брал максимальное возможное количество, и игрок Б легко выиграл, взяв на последнем ходе 19 спичек, на что имел полное право, так как $1 \leqslant 19 \leqslant 2 \times 16 = 32$. Конечно, это очень плохая стратегия. Вот другая партия:

Игрок А берет	Остается	Игрок Б берет	Остается
3	47	4	43
1	42	2	40
1	39	1	38
1	37	2	35
1	34	2	32
3	29	6	23
2	21	4	17
4	13	2	11
3	8		

Игрок А близок к победе. Если Б возьмет 3 спички и останется 5, то А имеет право взять последнюю. Это тем более верно, если Б возьмет более чем 3 спички (4, 5 или 6). Игрок Б не может взять больше и во всех случаях дает

игроку А все права, чтобы он мог взять последнюю спичку). Единственными случаями, подлежащими обсуждению, остаются поэтому случаи, когда Б берет одну или две спички.

Если Б берет одну, то остается 7, А берет 2 и остается 5. Если Б после этого берет более одной, то А берет последнюю; если Б берет одну и остается 4, то А берет одну и остается 3; Б может взять только одну или две, и кончает игру А.

Если Б берет 2 и остается 6, то А берет одну и остается 5, и А выигрывает тем же способом.

Я так подробно разбирал этот пример, чтобы познакомить вас с основными идеями. Тщательно изучите этот пример. Всегда ли игрок А заведомо выигрывает, как только он оставляет в куче 8 спичек...

Не предпринимайте здесь слишком глубокого изучения выигрывающей стратегии. Мы к ней еще вернемся ниже. Устройте только, чтобы ваш компьютер выигрывал при приведенных выше условиях, если старт для него благоприятен...

Вы легко сообразите, как представить эту игру на экране.

??** Игра 24. Гениальный отгадчик.

Вы уже сделали гениального ответчика; эта игра сложнее. Составьте программу для отыскания комбинаций, задуманных гениальному отгадчику. Вы можете идти двумя путями:

— вы выбрали комбинацию. Компьютер предлагает вам свою, затем читает число черных и белых шашек, которые получаются из того, что он вам предложил. Он должен найти ответ за наименьшее возможное число ходов;

— вы выбрали исходную комбинацию и сообщили ее компьютеру. Дальше все идет автоматически. Он выбирает некоторую комбинацию, определяет число черных и белых шашек, сообщает это все, затем переходит к следующей комбинации — пока не найдет ответ. Компьютер честен и не хитрит: он не использует того, что он знает задуманную комбинацию...

Вы скажете, что это неинтересно. Но это не так. Во-первых, стратегия поиска является вызовом для способности мышления. Мне пришлось немного подумать, чтобы получить в свое время разумный ответ с 6 позициями и 8 цветами. Попробуйте сами и убедитесь! С другой стороны, для гениального отгадчика существует проблема эф-

фективного начала. В программе, составленной мною, я сам выбираю первые испытательные комбинации. Я смотрел, сколько было систематических попыток и каковы они были. Это позволило понять, насколько важен начальный выбор и может ли он сильно влиять на результаты. Это — хорошее орудие экспериментирования. И это очень легко устроить. Компьютер запрашивает вас, сколько опытов априори вы хотите осуществить. Затем он запрашивает у вас начальные комбинации, число которых он только что прочел. После этого компьютер предпринимает систематическое исследование, какая из предложенных комбинаций должна быть оставлена.

Чтобы преуспеть, вам нужен хороший метод, и программировать нужно очень тщательно.

?** Игра 25. Погоня за сокровищем.

Любой начинающий в информатике мечтает сделать программу — чемпиона мира по шахматам... Я и сам видел несознательных, бросавшихся на эту задачу. Чтобы утешить их, нужно сказать, что это — одно из больных мест истории информатики. Компьютеры были еще электронными монстрами, напичканными радиолампами, которые приходилось охлаждать кубическими метрами воды, когда Герберт Симон (недавний Нобелевский лауреат по экономике) уже сделал примечательные предсказания:

— через 10 лет чемпионом мира по игре в шахматы станет компьютер, по крайней мере если правила не будут запрещать им участвовать в соревнованиях;

— через 10 лет компьютер обнаружит и докажет новую важную математическую теорему;

— через 10 лет большая часть диссертаций, выпускаемых по психологии, будет облечена в форму программ для компьютеров или качественных комментариев к примечательным особенностям компьютерных программ (Герберт А. Симон, Аллан Кьюэлл: Эвристическое решение задач: следующее продвижение в исследовании операций. «Operations research», т. 6, январь-февраль 1958, с. 6).

И через 25 лет с момента предсказания у нас не возникло проблемы запрещать доступ компьютеров к шахматным чемпионатам, они не представляют серьезной угрозы. Да, одна из программ выиграла партию у чемпиона (каждый человек имеет право ошибиться; конечно, это относится и к чемпиону. Он был очень усталым). Ни одна важная теорема машиной не обнаружена. Что же касается диссертаций по психологии, то, может быть, даже хорошо, что предсказание Симона не оправдалось... Очень боль-

но видеть, до какой степени информатика является благоприятным местом для ложных предсказаний. Сколько их нам обещали, этих чудес, которых мы так никогда и не увидели! Два года тому назад, во время Сикоба, журналист первой программы Французского телевидения показывал чудесную машину. Она была удивительно похожа на фотокопировальную. Он приподнял крышку, нашел там букву, нажал кнопку «и теперь буква зарегистрирована и мы ее легко узнаем, когда это потребуется». Фантастика! Покончим с этими мучительными сеансами поисков буквы этим господином, который два месяца назад подписал по этому делу контракт, номер которого я забыл... Но это был не господин, это была дама, и это было не два месяца назад, это был прошлый февраль. Больше никто не говорил об этой волшебной машине. Как же может случиться, что молодые люди не имеют никакого здравого смысла в информатике, так что можно без опаски предсказывать самые фантастические и самые неправдоподобные вещи, не вызывая ни смеха, ни краски стыда. «Мы подготовим 100 000 преподавателей за пять лет...» Но это — совсем другая история, как говорил Киплинг.

```
8 X • • • O • • •
7 • O • • • • O •
6 O • O • • • • •
5 • • O • O • • O
4 • O • • • O • •
3 • • • O • • • •
2 • • • • • • • O
1 • O • • • • • *
  A B C D E F G H
```

Рис. 15

Вернемся-ка к шахматам. Речи нет о том, чтобы вы ими занялись; это выше понимания любителя, даже самого талантливого. Но я хочу предложить вам нечто, что все же имеет какое-то отношение к шахматам и одновременно может позволить упражняться в постановке пьесы.

Я представляю шахматную доску на рис. 15 как на экране своего микрокомпьютера в виде квадратной таблицы с 64 полями, которые представлены точками. На шахматной доске в левом верхнем углу расположена черная ладья (помеченная крестом), а в нижнем правом углу — белая ладья (помеченная звездочкой). Тринадцать шашек, помеченных маленькими кружочками, случайным образом расположены на игровом поле. Компьютер перемещает черную ладью ×, а вы — белую ладью *. Каждый игрок на своем ходе передвигает ладью, как при игре в шахматы; только на поля на той же строке

или в том же столбце. Можно взять шашку и встать на место, которое она занимала; тогда эта шашка выходит из игры. Можно взять противоположную ладью, если оказывается возможным попасть на занимаемое ею место. Тогда игра останавливается, и тот, кто взял чужую ладью, и есть победитель. В противном случае игра останавливается, когда больше шашек нет. Тот, кто взял больше шашек, и есть победитель.

Вам необходимо указывать компьютеру, какой именно ход вы хотите сделать. Вы можете, например, отметить строки цифрами, а столбцы — буквами, как на рисунке. Ваш первый ход будет, без сомнения, на H2 или B1...

Стратегия совершенно не очевидна. У вас много возможностей. Не так много, конечно, как в шахматной игре, но достаточно для того, чтобы вам пришлось заняться всерьез, что бы написать программу, которую было бы трудно побить.

```
  .  .     .
  O  O  .  +
  +  O  O  +
.  +  +  O  +  .  .
1  2  3  4  5  6  7
```

Рис. 16

Если вы при этом достигли совершенства, почему бы не попробовать ее вариант, который не должен вызывать намного больше затруднений (???): та же задача, но ладьи заменены конями.

***Игра 26. Могущественная четверка.

Эта игра продается на рынке в другой форме. Она происходит в прямоугольном пространстве с 5 строками и 7 столбцами. Игра ориентирована, у нее есть низ и верх. Игровые позиции суть наинизшие свободные места в каждом столбце. Каждый игрок на своем ходе помещает свой отличительный знак на одно из игровых полей: например, один ставит крестики (+), другой — нолики (0). Первый, кто поставит на одной линии четыре принадлежащих ему знака — либо горизонтально, либо вертикально, либо по диагонали — выигрывает. На рис. 16 будем считать, что нолики при игре ставит тот игрок, чей ход именно сейчас. Если он не сыграет немедленно в пятом столбце, то его противник выиграет следующим ходом. По диагонали, начинающейся у основания четвертого столбца и идущей влево и вверх, есть три нолика, но единственное игровое поле в первом столбце обозначено точкой, и немедленно реализовать продолжение линии поэтому нельзя. Очевидно, что его противник не имеет никакого желания служить ему подставкой при пополнении первого столбца вместо того,

чтобы заниматься разыгрыванием мест, допускающих продолжение линии...

Эту игру, производную от вошек, программировать намного проще, потому что всего полей только 35, и только 7 из них являются игровыми полями на каждом ходе. Это существенно ограничивает работу. В реализованной мною версии ответ микрокомпьютера практически мгновенный (порядка секунды). Я не думаю, что я располагаю программой-чемпионом, я не очень хорошо знаком с этим родом игр...

5. СТРАТЕГИЯ БЕЗ ИГРЫ (ВЫИГРЫВАЮЩИЕ СТРАТЕГИИ)

Я объединил в этой главе несколько игр, которые можно найти на рынке и для которых существует стратегия решения. Как только она становится известной, игра теряет всякий интерес. Единственное связанное с такими играми удовольствие — обнаружить, как с ними покончить. Поэтому напишите программу — это наилучший способ сформулировать выигрышную стратегию, а затем забудьте игру, она вам больше ничего не принесет. И тем хуже, если продавцы этих игр не согласятся со мной... Некоторые из этих игр являются классическими среди информатиков. Я попытался их немного подновить. Многие стратегии могут быть элегантно запрограммированы с помощью рекурсивных процедур, но на языке Бейсик это невозможно. Всегда наступает день, когда фанатики этого языка, такого удобного для первых шагов, начинают понимать его ограниченность... Рекурсивность допустим в языках LSE и Паскаль.

? Игра 27. Бездельник.

Эта игра на рынке есть. Она имеет вид дощечки, в которую продето n гвоздей, скользящих через соответствующее отверстие, причем концы гвоздей расплющены и в каждом просверлено отверстие, в которое продето кольцо. Вы безусловно можете изготовить все это сами, используя достаточно толстые гвозди (диаметром порядка четырех миллиметров). Пропустите гвоздь в отверстие в 5 миллиметров в дощечке, а затем расплющите острие молотком. Просверлите головку наконечника, образовавшуюся у конца гвоздя, и вставьте туда кольцо для ключей. Каждое кольцо должно проходить вокруг предыдущего гвоздя. Трудность игры зависит от n. Для $n = 6$ она довольно быстро приходит к концу. Для $n = 8$

она требует долгих минут. Она почти невыполнима, если n больше восьми.

Через кольца проходит челнок, длинный замкнутый контур, представленный на рисунке. Дело в том, чтобы его вынуть и, таким образом, освободить от колец (рис. 17).

Первое, что нужно сделать — это научиться, как пропускать одно кольцо через челнок, или как его оттуда вынуть. Несколько манипуляций — и вы быстро убеждаетесь, что в какой стадии ни была бы игра, всегда можно надеть или снять первое кольцо, которое свободно (не проходит вокруг какого-либо гвоздя). Можно также

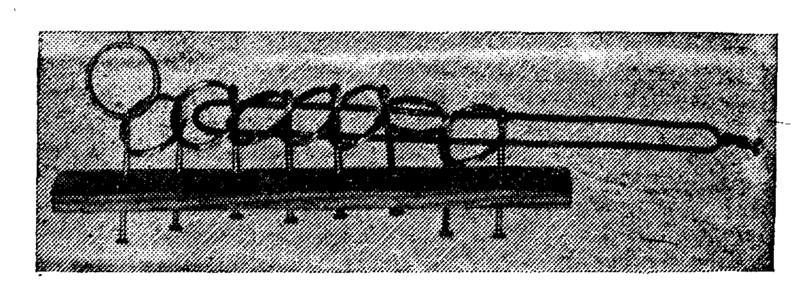


Рис. 17

освободить кольцо, которое следует за первым занятым кольцом (если оно проходит вокруг челнока), или одеть его на челнок, если оно не одето. Таким образом, игру «бездельник» можно заменить равносильной игрой, которую легче представить на компьютере.

Эта игра ведется на таблице, разделенной на несколько полей (8 полей на рисунке). В начальном состоянии каждое поле покрыто шашкой. Поля размечены цифрами. Играть на данном поле — значит, поставить туда шашку, если поле пусто, и удалить шашку, стоящую на этом поле — в противоположном случае. Правила игры следующие:

— можно всегда играть на первом поле,

— можно играть на поле, которое следует за первым занятым полем.

Есть две возможных игры:

НАДЕВАТЬ: игровое поле вначале пусто. Заполнить все поля.

СНИМАТЬ: игровое поле вначале наполнено шашками на каждой клетке. Нужно все убрать.

Эта задача имеет очень элегантное рекурсивное решение. Но если вы немного подумаете, то вы сможете также найти очень простое итеративное решение, причем игра НАДЕВАТЬ оказывается более простой, чем игра СНИМАТЬ.

Вот другая интерпретация этой игры — для тех, кто любит арифметику. Вы можете считать, что каждое поле может принимать два состояния (свободное и занятое), что эквивалентно двоичным числам — например, 0 для свободного и 1 для занятого полей. Тогда каждая конфигурация является представлением целого числа по основанию 2. Таким образом, рис. 18 представляет целое число 11111111 в качестве начального состояния и 01011011 в качестве промежуточного состояния. Ниже нам будет удобно читать эти слова в обратном порядке, так что в этих новых обозначениях промежуточное состояние соответствует двоичному числу 11011010.

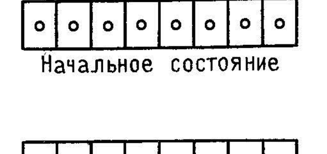
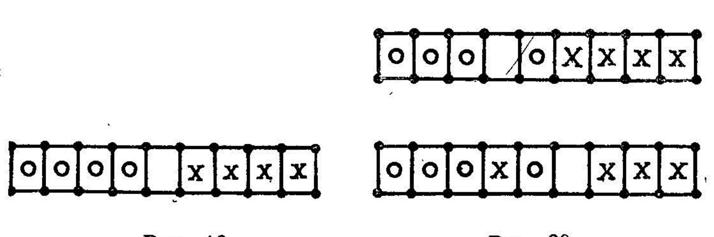


Рис. 18

Ясно, что эта игра порождает последовательность чисел (в приведенном выше примере число равно 218 в десятичной записи). При переходе от одного числа к следующему меняется лишь одна двоичная цифра. Можете ли


Рис. 19 Рис. 20

вы сказать, какая последовательность порождается таким образом в каждой из игр?

?* Игра 28. Зануда.

Эта игра называется также «игра в лягушек». У нее была версия, использованная в материалах лицеев, но в ней было не все, что я вам сейчас предлагаю. Игровое поле снова имеет вид прямоугольной площадки, разделенной на поля. Число полей должно быть нечетным (9 на

рис. 19). Поля слева покрыты шашками некоторого цвета (я представил их ноликами), поля справа — шашками другого цвета (здесь — крестиками). Среднее поле свободно. Крестики могут передвигаться только влево, нолики — только вправо. Шашка может быть либо подвинута на один шаг, если следующее поле в направлении ее перемещения свободно, либо перепрыгнуть через шашку другого рода, если следующее за ней поле свободно. Рисунок 20 иллюстрирует два возможных хода в партии с начальным положением на рис. 19.

Цель игры состоит в том, чтобы привести все X влево, а все 0 вправо, так что конечное состояние должно быть похоже на начальное, и шашки должны поменяться местами (крестики справа, нолики слева).

Программа, которую вы должны составить, должна описывать последовательность перемещений шашек для произвольного (но, конечно, нечетного) числа полей. Вы можете получить решение в виде пары рекурсивных процедур или в виде одной итеративной программы. Как только вы найдете стратегию, зануда не будет больше представлять никакого интереса. Как это случилось с теми, с кем я занимался на Митра 15, в лицее, требуя, чтобы игрок сидел за своей клавиатурой и переставлял шашки. Но если не знать стратегии и действовать случайным образом, то выиграть нельзя вследствие теоремы Дюнойе: «Если какой-то выбор вы делаете случайным образом, то вы всегда проигрываете». Это нам постоянно повторял наш учитель математики, когда я был в подготовительном классе Политехнической школы. Мы придумали следствие: поскольку мы всегда проигрываем при случайном выборе, то достаточно после этого выбора выбрать другую сторону альтернативы. Но это дает выход из парадокса Дюнойе (я совершенно не знаю, кто такой Дюнойе. Это — существенный момент истории науки, который следовало бы прояснить. Всегда цитируют Мэрфи и его знаменитые законы: если в некотором опыте что-то может разладиться, то можно быть уверенным, что это обязательно произойдет. Если, кроме того, при этом в комнате есть посторонний наблюдатель, то он прибавит «ну, я же так и говорил...». Дюнойе — предшественник Мэрфи). Вот в чем парадокс. Есть альтернатива. Вы выбрали случайным образом и обманулись. Следовательно, если вы взяли другую сторону альтернативы, то вы оказались правы. Но это — тоже случайный выбор, поэтому вы опять обманулись...

?* Игра 29. Б — А — БА.

Эта игра вовсе не потому самая простая среди всех игр этого сорта, что она называется б—а—ба. Согласно [BAL], она имеет японское происхождение. Ее можно сформулировать следующим образом. Игра разыгрывается на площадке, разделенной на клетки, на этот раз в четном числе. Есть шашки двух сортов, скажем, крестики и нолики — как в «заиуде». В начале игры два левых поля свободны, остальные заняты поочередно 0 и ×, как указано на рис. 21. При каждом ходе вы можете переместить пару смежных шашек, перенося ее на пару смежных свободных клеток. Вы выиграете, когда все × будут вместе стоять на левых полях, затем будут нолики, а два правых поля останутся свободными.

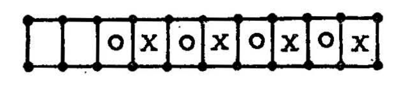


Рис. 21

Можно также представить это другим способом. Свободные поля представляются точками (рис. 22), остальные заняты буквами *а* и *б* (вот вам и *б — а — ба*).

. . а б а б а б а б	Начальное состояние
б б б б а а а а . .	Конечное состояние
б а а б а б а . . б	Возможный первый ход

Рис. 22

Пара шашек, которая переносится при данном ходе, абсолютно произвольна: две одинаковых буквы, две разных буквы, все равно в каком порядке...

Начните с решения задачи для 8 букв и 10 полей, как на рисунке. Это очень просто и у вас нет необходимости в компьютере. Попробуйте затем решить ее для бóльшего числа полей.

Честно говоря, я соответствующую программу не написал, потому что ее использование на компьютере меня ничему новому не научило бы. То, что здесь приведено, подходит программисту, который на что-то рассчитывает. Если у вас есть склонность к программированию, то вы найдете способ решить задачу для всех случаев.

* Игра 30. Отшельник.

Может быть, мне и не следовало бы помещать «Отшельника» в эту главу. Классификация игр полностью основана на оттенках и на индивидуальных оценках.

Я провел немало времени в забавах с «Отшельником», но все же верно, что едва только удается обнаружить хорошую стратегию, как интерес уменьшается. Возможность его программирования связана с улучшениями. «Отшельник» разыгрывается на площадке с проделанными в ней отверстиями, в которых могут быть размещены шашки. Но можно также использовать доску, на которой нарисованы поля, а можно также все это очень хорошо нарисовать на земле и использовать камешки в качестве шашек — точно так же, как я рассказывал при игре в лис и кур. Впрочем, может оказаться, что «отшельник» был изобретен каким-нибудь знатным французом, заключенным в Бастилию, который модифицировал игру в лис и кур.

```
    • • •
    • x •
• • x x x • •
• • • x • • •
• • • x • • •
    • • •
    • • •
```

Рис. 23

Рисунок 23 представляет одно из состояний игры в «Отшельника». Свободные места представлены точками, шашки — знаком ×. Как показывает название, это — игра для одного-единственного лица. При каждом ходе нужно съесть шашку, заставляя перепрыгнуть через нее другую шашку так, чтобы попасть на свободное поле — либо горизонтально, либо вертикально. Так, на рис. 23 имеется 4 возможных хода:

— шашка, лежащая на пересечении планок креста, может взять шашку, расположенную непосредственно над ней, и попасть в середину верхней строки (шашка, через которую перепрыгнули, а именно, расположенная в вершине креста, удаляется из игры);

— та же шашка может взять шашку слева;

— или шашку справа;

— наконец, шашка в центре игрового поля может взять шашку под ней, расположенную в низу креста.

Цель игры состоит в том, чтобы удалить все шашки, кроме одной. Число необходимых для этого ходов легко подсчитать: поскольку при каждом ходе берется одна шашка, то число ходов равно числу подлежащих удалению шашек. В случае креста на рис. 23 вам осталось сделать до конца еще 5 ходов.

Составьте программу для отшельника. Вы даете компьютеру начальную конфигурацию, например крест на рис. 23. Он сообщает вам ходы, которые приводят к решению.

Другие конфигурации приведены на рис. 24.

В своей наиболее общей форме эта игра начинается с игрового поля, полностью покрытого шашками, кроме единственного остающегося свободным поля. Попробуйте

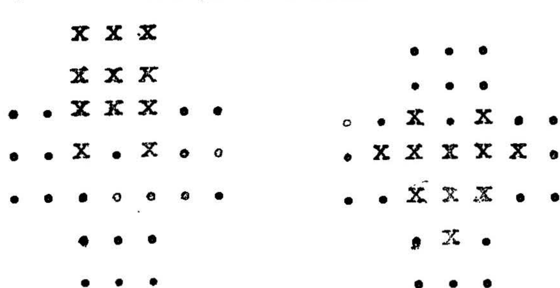


Рис. 24

заставить вашу программу работать и в этом случае. У вас появится новая трудность, связанная с симметрией игры: есть много решений, эквивалентных с точностью до симметрии.

```
    x x x                 x x x
    x x x                 x . x
x x x x x x x         x x x . x x x
x x x . x x x         x x x x x x x
x x x x x x x         x x x x x x x
    x x x                 x x x
    x x x                 x x x

              x x x
              x . x
          x x x x . . x
          x x x x x x x
          x x x x x x x
              x x x
              x x x
```

Рис. 25

На рис. 25 сначала изображена исходная конфигурация. Есть 4 различных хода — шашка из перекладины креста прыгает в центр игрового поля. Эти 4 хода начи-

нают 4 решения, эквивалентных с точностью до поворота на прямые углы. После этого остаются еще две возможности эквивалентности с помощью симметрии относительно вертикальной оси игры. Нижняя конфигурация показывает результат одного из этих ходов.

В позиции, к которой мы пришли, никакой симметрии уже нет. Ее-то и возьмите как исходную.

Ваша программа дает только одно решение или все возможные решения?

Ханойские башни. Печальный конец Паскаля Младшего

Очень мало говорят о печальном конце Паскаля Младшего. Его бывшие коллеги знали, что у него возникли проблемы, заставившие поместить его в психиатрический госпиталь. Теперь когда он умер, я могу опубликовать письмо, которое он мне послал в свое время; оно уже больше не может причинить ему вреда...

«Господин профессор,

Я не знаю, помните ли вы меня: я был вашим учеником в Институте программирования. Конечно, у вас их столько было... После того, как я окончил институт, я поступил на работу программистом-аналитиком в бюро обслуживания. Я был на очень хорошем счету. Я следовал вашим урокам: использовал программирование «сверху-вниз», я выводил свои циклы в программах, используя пост- и предусловия и инварианты. Мои программы работали верно с первого запуска, с точностью до опечаток. Короче, по прошествии нескольких лет я сказал себе, что у меня будет более интересная работа, если я буду вести ее на свой собственный счет. Поэтому я все подготовил, нашел помещение. Я подал в отставку и взял все отложенные отгулы, на которые я имел право. Будучи холостым, я, вообще говоря, брал очень мало выходных дней, настолько меня захватывала моя работа. Но, собираясь испытать счастья в большом деле и становясь своим собственным работодателем, я хотел получить настоящий отдых.

Право, я не знаю, как это меня по рекламному объявлению занесло в бюро путешествий «Посетите таинственную Индию». И вот я отправился с тремя десятками других в организованное путешествие. Конечно, я должен был задуматься раньше, то ли я выбрал, что мне нужно. Оказалось, что я с трудом переношу беспрерывную болтовню то одних, то других; это мешало мне думать о чем-

нибудь своем. Мне пришлось примириться с тем, что мне придется думать о чем-то еще, кроме написания какой-то упирающейся программы! Детали этого путешествия несущественны вплоть до дня, когда нас привезли в монастырь в предгорьях Гималаев.

Монах, который нас принял, говорил на отличном французском. Сообщение, которое он сделал о монастыре, свидетельствовало о свободном владении нашим языком. Это должно было показаться мне подозрительным. Он ввел нас в помещение и, с того момента как я вошел

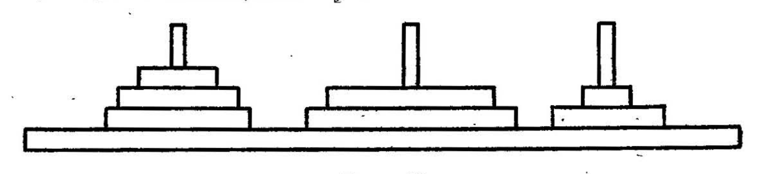


Рис. 26

я смог только выдавить «ох» изумления: мы увидели монаха, который занимался знаменитейшей игрой в Ханойские башни. Диски были, очевидно, из золота, и я сразу угадал, даже не считая, их ровно 50. Монах объяснял игру посетителям:

«Как вы видите, игра состоит из круглой подставки с тремя стержнями. На стержни нанизаны диски различных диаметров. (Рис. 26 представляет конфигурацию игры с семью дисками.) На каждом стержне диски сложены в стопку по возрастанию диаметра: никогда ни один диск не кладется на другой диск меньшего диаметра. В начале игры, это было много лет назад, даже много веков, все пятьдесят дисков находились на первом стержне. Монах перекладывает эти диски один за другим, следуя методу, который мы тщательно продумали. Когда все диски соберутся снова на одном стержне, отличающемся от исходного, игра кончится. Великий труд, который боги наложили на людей, будет завершен, и сможет наступить конец света...»

Я решил насмешливо добавить вполголоса: ну, это еще не завтра... Это стоило мне разгневанного взгляда гида, который продолжал:

«Как только что заметил один из вас, это занятие потребует еще многих столетий, несмотря на большую сноровку монахов, которые перекладывают каждый диск приблизительно за одну секунду». Я снова продемонстрировал свое раздражение. Разумеется, я предполагал, что

из-за меня посещение будет сокращено. Но не для того же я сюда приехал, чтобы надо мной, как и над остальными путешественниками, насмехались? Мы хорошо знаем, что игра в ханойские башни была изобретена в конце прошлого века преподавателем математики в лицее Сент-Луи по имени Люка, который под этим соусом ее и пустил в свет, окружив легендой, согласно которой монахи где-то в Индии суетятся вокруг игры в 50 дисков, по окончании которой наступит конец света. Эта легенда делала естественной задачу о подсчете числа ходов, необходимых для завершения игры. Что же касается того, что каждый диск перекладывается за секунду, то это элементарно, и мы знаем итеративную стратегию, которая позволяет нам просто играть, ни о чем не думая — вы ее нам сами давали в вашем курсе в институте...

Когда мы покидали монастырь, наш гид подошел ко мне и спросил меня, не хочу ли я оказать его настоятелю большую честь своим посещением. Обсуждение с руководителем группы. Назавтра мы не должны были уезжать рано, и поэтому я принял приглашение снова прийти туда до нашего отъезда. Настоятель принял меня очень любезно и предвосхитил мои упреки: «Несомненно, вы уже знакомы с башнями Брахмы. Мы знаем, что они были введены во Франции много лет назад М. Люка. Он никогда не говорил, что он сам придумал зту игру. Совсем наоборот, он очень добросовестно изложил то, что мы делаем. И разве мы виноваты в том, что вы вбили себе в голову, что с его стороны это была чистейшая уловка, чтобы придать бóльший блеск своему мнимому открытию? А зто была и в самом деле чистейшая уловка, потому что ему приписали создание этой игры, в то время как он всего лишь пересказал то, что ему описал один путешественник... Нас тревожит то невероятное время, которое нужно для окончания игры. Мы очень терпеливы, однако мы ищем, как двигаться быстрее. Один наш посетитель, приехавший из американского университета, предложил нам сконструировать робота-манипулятора, управляемого компьютером. Мы со своей стороны финансировали это исследование. Но робот не мог двигаться быстрее, чем наши монахи, натренированные до совершенства и действовавшие безошибочно». Когда же я высказал замечание, что при таком решении проблемы игру будут вести уже не люди, а машина, настоятель решительно возразил мне, сказав: «Мы прекрасно пользуемся молитвенными мельницами. Во всяком случае, машина делается

людьми и управляется программой, написанной людьми...»

Он также сказал мне, каким образом это исследование к тому же открывает новые перспективы. Были времена, когда монахи пытались присоединить к игре четвертый стержень. Правила оставались такими же: перемещать за один раз не более одного диска и никогда не класть диск на другой диск меньшего диаметра. Конечно, манипуляция игрой с 50 дисками до сих пор не удалась. Они вывели, что при этом требуется гораздо меньше ходов, но стратегия манипулирования становится много сложнее. Монахи терялись, часто оказывалось, что они ошибаются, они снова попадали в уже пройденные конфигурации, так что не было никакой уверенности в том, что удастся дойти до конца, если постоянно приходится начинать сначала... «Не могли бы вы взяться за решение проблемы башен Брамы с четырьмя стержнями, составить программу для соответствующего компьютера и использовать его для управления роботом, манипулирующим игрой? Ведь даже если каждый ход отнимет много секунд, конец должен будет наступить намного быстрее. А нам, таким образом, выпадет величайшая радость — стать теми, кто выполнил волю богов. Мы увидим, что мир достиг своего конца, и вступим в счастье, которое никогда не кончится...»

Это дело показалось мне выполнимым. Договорились, что я реализую информативную систему и передам ее им. Настоятель вручил мне в качестве оплаты авансом игру, сделанную из серебра. Это было настоящее богатство. Ну, как тут устоять?

Вернувшись во Францию, я взялся за работу. Больших трудностей она не представляла. Вначале я составил рекурсивную процедуру для решения игры с четырьмя стержнями. Поскольку я искал оптимальную стратегию, я сделал по ней итеративную версию. Для этой маленькой программы был достаточен микрокомпьютер. Я использовал ручной манипулятор, оснащенной электромагнитом в форме кольца. Я работал с деревянной игрой, каждый диск которой был снабжен маленьким кольцом из мягкого железа, позволявшим ручному манипулятору брать его, и притом не возникало необходимости чрезвычайно точно этот диск устанавливать.

И я был всем очень доволен, пока не понял внезапно, что я без раздумий бросился в ужасное предприятие. А что, если монахи говорили правду? Какую пользу

мне принесет обладание богатством (ибо они должны были заплатить мне еще, если программа будет работать правильно), если вскоре наступит конец света? Безусловно, будучи убежденным рационалистом, я не очень-то всерьез принимал их истории. Но, в конце концов, это — новая форма пари Паскаля *). Даже если шанс, что все это верно, бесконечно мал, я не испытывал ни малейшего желания ускорять конец света. Но прошлое вернуть нельзя, и их плату я уже получил.

Мне пришла в голову поистине дьявольская мысль: эти прекрасные монахи желали конца света, чтобы как можно скорее достичь вечного счастья, вот уж я его им обеспечу. В корпус компьютера я добавил выдвижной ящик, который окрестил «концом света». В нем были под видом блока питания толстые цилиндры, на корпусах которых была маркировка конденсаторов, но в которых находились пластиковые бомбы. Маленькое изменение программы должно было вызвать взрыв сразу же после того, как наименьший диск покидал свой стержень и перед тем, как он достигал места своего назначения. Таким образом, игра никогда не должна была кончиться. Что до монахов, то они будут с восторгом представлять себе конец света в тот момент, когда завершится игра. В тот момент чудовищность моего поступка меня не шокировала. Наоборот, я был в восторге: монахи будут счастливы, а я уберегу весь мир от конца. Я дошел до того, что смотрел на себя как на благодетеля человечества. Конечно, я брал на себя риск. Программу, без сомнения, нужно было испытать. Но, как я вам уже говорил, я прошел хорошую школу — Вашу школу — и я программировал правильно.

Когда все было закончено, я отправился вручить свое произведение сияющим монахам. Она была испытана на игре в 20 дисков. Затем аппарат был пущен в работу для игры с 50 дисками и 4 стержнями. Тут я попросил у монахов разрешения удалиться: ведь мне хотелось бы привести свои дела в порядок в то небольшое время, которое

---

*) Имеется в виду постановка Блезом Паскалем (1623—1662) вопроса о вере в существование бога как задачи о выборе стратегии в азартной игре («Мысли», отрывок 233): «Взвесим выигрыш и проигрыш, ставя на то, что бог есть. Возьмем два случая: если выиграете, вы выиграете все; если проиграете, то не потеряете ничего. Поэтому, не колеблясь, ставьте на то, что он есть» (Антология мировой философии в четырех томах, Том 2, М., «Мысль», 1970, с. 306).— *Примеч. пер.*

осталось нам жить, что они очень хорошо понимали. Я возвратился с полными пригоршнями золота.

Через некоторое время сообщили, что ужасный взрыв неизвестного происхождения разрушил монастырь в Индии. В живых не осталось никого. Моя программа была правильной...

Уже позже меня стали одолевать сомнения. Не были ли монахи правы? Не я ли тот, кто помешал воле богов? Не воспрепятствовал ли я выполнению работы, которая была доверена людям? Не должен ли я сконструировать игру в 50 дисков, и не следует ли ее разыграть, чтобы искупить свою вину?

С этих пор я живу в ужасе. Если я ничего не делаю, я несу на себе груз того, что я препятствую воле богов. Если я сделаю игру, что для меня не составляет никакого труда, то именно я и приведу мир к гибели... Я никому не могу довериться. Я умоляю вас, помогите мне...»

Я не стал вмешиваться. Душа Паскаля Младшего не могла сопротивляться этому удару. Он впал в безумие и немного спустя умер...

Игра 31. Рекурсивная форма.

Письмо Паскаля Младшего ставит много задач по программированию. Я не осмеливаюсь предложить вам написать рекурсивную процедуру, которая перечисляет последовательность движений дисков в игре с тремя стержнями, помеченными номерами 0, 1 и 2 (например, в форме последовательности строк ДИСК 2 ИДЕТ С 1 НА 0, дающих номер перемещаемого диска, если наименьший диск имеет номер 1, номер стержня, с которого диск снимается, и номер стержня, на котором этот диск оказывается).

Тем не менее эта процедура была бы для вас очень полезна как для наблюдения за перемещениями в течение партии, так и для изучения свойств игры.

Можете ли вы вычислить число $f(n)$ ходов, необходимое для проведения партии в игре с n дисками? Сколько веков потребуется для проведения игры в 50 дисков, если каждый ход делается за секунду?

Я использовал игру из дерева, в которой диски были обтесаны из двух разных пород дерева, поочередно светлых и темных. Проводя игру, можно убедиться, что два диска одного и того же цвета никогда не оказываются друг на друге. Сумеете ли вы показать это с помощью рассуждения, основанного на рекурсивной процедуре? Заметьте, что это сводится к вопросу четности. Если

диски занумерованы так, как это было описано выше, то диски с номерами одинаковой четности никогда не попадают друг на друга.

* Игра 32. Рисунок игры.

Напишите простую рекурсивную процедуру, наиболее образно дающую возможность увидеть движение дисков. Очень общий способ состоит в том, чтобы изобразить три стержня в виде трех строк, на которых последовательно поставлены номера дисков. Таким образом, рис. 27 представляет начальное состояние и промежуточное состояние игры с 5 дисками.

0	5 4 3 2 1	0	5 1
1		1	3 2
2		2	4

Рис. 27

Внимание: ваша программа работает слишком быстро, и вы не видите перемещений. Вставьте цикл ожидания, чтобы замедлить игру...

Более хитрый способ представляет стержни вертикально либо как последовательность номеров, либо — что еще лучше — если у вас есть графическая система, стилизованным образом, как на рис. 26. Это труднее...

? Игра 33. Итеративная стратегия.

Таких стратегий много. Сумеете ли вы предложить такую, которая позволила бы играть по ходу в секунду, как у монахов...

??? Игра 34. Игра с 4 стержнями.

Составьте рекурсивную программу для игры с 4 стержнями, не занимаясь представлением дисков на экране каким-либо хитрым образом. Вам и без этого будет достаточно работы. Даже если ваш компьютер не предоставляет вам возможности вычислять рекурсивные процедуры, это поможет вам ясно увидеть задачу и найти стратегию.

Найдите способ действовать, минимизируя число ходов, и найдите их необходимое число для малых значений n. Из письма Паскаля Младшего неясно, сколько времени нужно — если каждый ход совершается за секунду — для завершения игры с 4 стержнями и 50 дисками. Вычислите его.

??** Игра 35. Составьте итеративную программу для игры с 4 стержнями.

Если теперь добавить пятый стержень, то нужно все начинать сначала, или результаты, полученные для 4 стержней, допускают немедленную экстраполяцию на случай 5 стержней?

??? Игра 36. Спички Бергсона.

Эта игра была предложена выше (игра 23). Тогда требовалось только дать компьютеру указание, что он должен будет сделать, чтобы выиграть наверняка, если исходить из 50 спичек. Теперь нужно заглянуть глубже и изучить выигрывающую стратегию в полной общности.

Как и во многих играх, есть ситуации, которые позволяют игроку выиграть наверняка (если он их знает), и другие ситуации, исходя из которых выиграть невозможно. Пусть П обозначает ситуацию, благоприятную первому игроку (или предыдущему. П — это первая буква слова «позитивный», как это заметил Роуз Болл [BAL]). Эта ситуация хороша для меня, если это такая ситуация, которой я достигаю после своего хода. Ситуация Н благоприятна второму, «новому», игроку (неблагоприятна первому, негативна). Она хороша для меня, если это — та ситуация, которую я застаю в момент своей игры. Вся моя стратегия есть переход от ситуации Н в ситуацию П.

Это все работает, если, исходя из ситуации Н, я всегда могу достичь ситуации П с помощью разрешенного хода, и если, с другой стороны, налицо ситуация П, то я не могу достичь никакой ситуации П с помощью дозволенного мне хода. Итак:

— из ситуации Н всегда можно достичь ситуации П,
— из ситуации П можно достичь только ситуаций Н,
— выигрывающая ситуация есть ситуация П.

Игра происходит переходами между ситуациями Н и П. Победитель определяется природой — принадлежностью классу Н или П — начальной ситуации и, таким образом, определяется тем, кто начинает.

В игре в спички Бергсона ситуация характеризуется двумя целыми числами p, q:

p — число спичек, оставшихся в куче;

q — число спичек, взятых на предыдущем шаге. Тогда можно взять от 1 до $2q$ спичек.

Ситуация $0, q$ — выигрывающая для любого q.

Ситуации $1, q$ и $2, q$ суть ситуации Н. Исходя из них, можно взять последнюю спичку. Ситуация 3, 1 есть П: исходя из нее, нельзя получить ничего, кроме 2, 1 и 1, 2, и обе эти ситуации относятся к классу Н.

Составьте программу, перечисляющую положения П вплоть до некоторого уровня.

Понаблюдайте за результатами. Попытайтесь узнать их свойства и, если вам хватит смелости, проверить их.

Вы наверняка знаете числа Фибоначчи: они определяются рекуррентным соотношением

$$f(n) = f(n-1) + f(n-2)$$

и единичными значениями двух первых чисел. Какой черт их сюда занес...

6. КОМБИНАТОРНЫЕ ЗАДАЧИ

Я объединил здесь различные головоломки, решение которых для компьютера в принципе нетрудно. Есть конечное число возможных случаев. Мы испытываем их все и выбираем наилучший. К этой категории относится и чемпион мира по шахматам: конечное число шахматных фигур, конечное число правил, конечное число ходов...

Но, конечно, есть препятствие — это число опытов, которые нужно провести. Прежде всего нужно хорошо организоваться, чтобы некоторых попыток и не предпринимать. Нужно выбрать также путь, который предоставляет наиболее возможные шансы дойти до конца, и не вступать на такой путь, который не дает ничего, кроме неудач. В этом — все искусство программирования.

Головоломка 20. Восемь ферзей.

Возьмите на заметку: это самая простая головоломка подобного типа. Поставить на шахматной доске 8 ферзей так, чтобы они друг другу не угрожали. Ферзь может взять все, что находится в этой же строке, что и он, в том же столбце или на той же диагонали. Представим, как обычно, шахматную доску квадратной таблицей полей, среди которых свободные поля помечены точками, а ферзи помечены крестиками (×).

На рис. 28 представлены две попытки решения. На левой доске поставлены 4 ферзя. Все поля строки 6 ими уже блокированы. Продолжать дальше бесполезно. На правой доске мы сумели поставить 7 ферзей, но восьмая строка блокирована.

Я обратил ваше внимание на эту задачу, поскольку ее решения всюду приведены. Ее можно в высшей степени элегантным образом решить с помощью рекурсивной процедуры. Но нетрудно дать решение и в итеративной форме.

Деликатный вопрос связан с представлением шахматной доски. Но и возможности этого выбора также обсуждаются в известных книгах. Что же тогда остается найти?

Если у вас нет этих книг, то остается найти решение. Нет — решения, поскольку их 92. Но не все они существенно различны, так как шахматная доска обладает симметриями.

Поэтому пытайтесь искать только основные решения, исходя из которых и используя симметрии шахматной доски, можно найти все остальные решения...

```
1  x . . . . . . .    x . . . . . . .
2  . . x . . . . .    . . x . . . . .
3  . . . . x . . .    . . . . x . . .
4  . x . . . . . .    . . . . . . x .
5  . . . . . . . .    . x . . . . . .
6  . . . . . . . .    . . . x . . . .
7  . . . . . . . .    . . . . . x . .
8  . . . . . . . .    . . . . . . . .
```

Рис. 28

В этом примере вам не следует бояться сложности. Даже самые плохие программы будут все еще достаточно быстры...

?** Головоломка 21. *X* ферзей.

Поставить на шахматной доске 8 ферзей так, чтобы они друг другу не угрожали, можно. Но трудности, с которыми мы встретились при попытке достичь решения без помощи компьютера, ясно показывают, что нет необходимости в 8 ферзях, чтобы блокировать всю шахматную доску.

Каково наименьшее число ферзей, необходимых для блокирования шахматной доски, так, чтобы не было возможности поставить ни одной фигуры ни на одно поле, чтобы один из уже стоящих ферзей не мог эту фигуру взять?

Так как я вам не задал x, то вам нужно пытаться заставить x либо расти, либо уменьшаться. Впрочем, в этой задаче наши дела идут хуже, чем с 8 ферзями. В предыдущей задаче мы знали, что в каждой строке и каждом столбце обязательно должен быть ферзь. Если ферзей меньше 8, то это уже неверно.

* Головоломка 22. Домино.

Маленькая прелестная головоломка, совсем не трудная. Она была предложена на испытании на проницательность на конкурсе организации АФСЕТ по программированию в 1981 году.

Берутся 7 костяшек из одного набора домино. Напомним, что эти шашки сделаны из двух частей, на каждой из которых либо ничего не написано (чистая сторона), либо очки в числе от 1 до 6.

Задача состоит в том, чтобы образовать из этих 7 костей все возможные цепи, состыковывая костяшки домино частями с равными количествами точек. Нет никакой уверенности, что такая цепь существует.

Не ведите себя так, как некоторые из соревнующихся на этом конкурсе. Я тогда входил в жюри. Мы должны были оценивать работы соревнующихся. Если бы я принимал решения единолично, я потребовал бы, чтобы мне были представлены тексты программ, и я бы судил по самим произведениям. Но другие члены жюри нашли более длинный и более сложный метод. Они приготовили специальные тесты. Они должны были быть испытаны на программах соревнующихся, и нужно было подсчитать число правильных ответов, чтобы расклассифицировать соревнующихся. Новое обсуждение: я выдвигаю оценку, что и один-единственный неверный ответ выражает ошибочность программы и, следовательно, выводит ее из конкурса. В конце концов было решено, что так и будем делать. Все программы, содержащие ошибку, будут рассматриваться как неверные. Если две команды получат одинаково верные ответы, то мы еще раз детально изучим полученные результаты, стараясь разгадать природу ошибки при переходе к данному тесту от уже удавшихся, чтобы отдать одному из них предпочтение. Вот нам и досталось: один из соревнующихся, думая, что с удавшимися тестами это согласуется, пытался упростить программу для домино. Он сказал себе, что вне всякого сомнения, будут даны кости, из которых никаких цепей ставить нельзя. Его программа читала последовательность костей домино и сообщала НЕВОЗМОЖНО без каких-либо других вычислений. Если бы я не настаивал на своем так решительно, то он был бы не хуже других...

Не поступайте так. Эта задача при всем том нетрудная... Рис. 29 дает пример цепи.

* Головоломка 23. Последовательность 0—1—4—6.

Это — головоломка, на которую я натолкнулся, работая над своей диссертацией на ученую степень по физике. Я занимался сетями антенн для радиоастрономии. Сеть антенн состоит из основания, на котором по одной линии размещены отдельные антенны, доставляющие информацию о наблюдаемых нами звездах. Каждая пара антенн дает информацию о некоторой величине, пропорциональной расстоянию между двумя антеннами этой пары. Нас интересуют значения этой величины, образующие арифметическую прогрессию. Таким образом, нужно

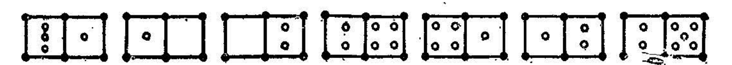


Рис. 29

было располагать антенны таким образом, чтобы расстояния между разными парами образовывали арифметическую прогрессию. Я предложил систему из 4 антенн, расположенных на прямой в точках с абсциссами 0 1 4 6.

Тогда получаемые из них 6 различных пар приводят к расстояниям между антеннами, пропорциональным следующим числам:

0—1	1
4—6	2
1—4	3
0—4	4
1—6	5
0—6	6

Можно сформулировать задачу по-другому. Нужно найти последовательность натуральных чисел $a_1, a_2, \ldots, a_n$ — последовательность, которую можно предполагать возрастающей — такую, чтобы попарные разности членов этой последовательности $a_j - a_i$ $(j > i)$ были попарно различны и образовывали последовательность всех целых чисел от 1 до $n(n-1)/2$.

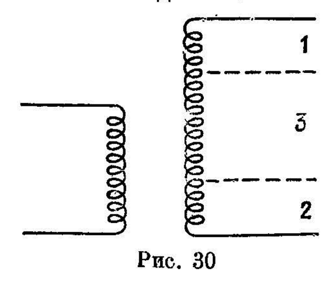


Рис. 30

Это — еще и проблема трансформатора (см. рис. 30). Если включить во вторичную обмотку 4 выхода так, чтобы число витков между первым и другими выходами

находилось в отношениях 1, 4 и 6, то можно получить 6 напряжений на выходе, образующих арифметическую прогрессию.

Опустим другие физические задачи, порождающие такие последовательности. Четырехчленная последовательность 0—1—4—6, по-видимому, является наибольшей последовательностью, обладающей свойством порождать последовательность первых целых чисел, не пропуская и не повторяя дважды ни одного из них, при попарном вычитании членов этой последовательности.

Так, для 5 целых можно образовать 10 разностей. Поэтому крайние члены должны быть $a_1 = 0$, $a_2 = 10$. Чтобы получить в виде разности 9 из двух членов последовательности, нужно, чтобы либо было $a_2 = 1$, и тогда $a_5 - a_2 = 9$, либо $a_4 = 9$. Эти два решения легко получаются одно из другого операцией симметрии. Поэтому положим $a_2 = 1$.

К этому моменту мы получили уже $a_1 = 0$, $a_2 = 1$, $a_5 = 10$. Чтобы получить разность, равную 8, нужно взять

— либо $a_3 = 2$, но тогда разность, равная 1, получается дважды:

$$a_3 - a_2 = a_2 - a_1,$$

— либо $a_4 = 8$,

— либо $a_4 = 9$, но тогда снова дублируется разность 1. Следовательно, $a_1 = 0$, $a_2 = 1$, $a_4 = 8$, $a_5 = 10$.

Достаточно теперь пересмотреть одно за другим возможные значения a_3 и удостовериться, что каждое из них дублирует какую-то разность.

Для a_3, равного 2, дублируется разность 1:

3	2
4	4
5	5
6	2
7	1

Таким образом, нет последовательности из 5 целых, попарные разности которых порождают 10 первых натуральных чисел. Допустим теперь возможность повторений в последовательности разностей. Можно ли с помощью 5 членов породить — с помощью их разностей — 9 первых натуральных чисел?

Об этих последовательностях ничего не известно. Пусть дано число n членов последовательности; каково наиболь-

шее число последовательных целых чисел, начиная с 1, которые можно получить с помощью попарных разностей членов последовательности?

Запрограммировать это не очень трудно. Но берегитесь чересчур долгих вычислений!

?** Головоломка 24. Прогулка королевы.

Нет, не в Булонском лесу, если говорить серьезно... Прогулки фигур на шахматной доске — классический сюжет для головоломок. Эйлеровский конь должен обойти всю шахматную доску и вернуться на поле, с которого отправился в путь, не попадая дважды ни на одну клетку. Это настолько общеизвестно, что это уже и не головоломка. Но если вы не знаете решения, я не мешаю вам попробовать.

Случай «королевы» (ферзя) — немного другой. Эта фигура может перемещаться по горизонтали, по вертикали или по диагонали. Назовем «движением» перемещение на некоторое число полей в определенном направлении. Разрешается много раз проходить по одному и тому же полю. Но требуетс пройти все поля за наименьшее возможное число движений, причем, конечно, нужно вернуться на исходное поле. Так как число движений не дано, то не попытаетесь ли вы сначала проделать все вручную, чтобы узнать верхнюю границу...

???* Головоломка 25. Девушки из пансиона Киркмана.

Пансион Киркмана — это колледж для девушек из высшего общества. Надзирательницей там — мисс Фармер. Каждую среду после полудня она сопровождает класс на прогулку. В своей нарядной униформе, в соломенных шляпках они медленно вышагивают по трое в три ряда. Мисс Фармер несговорчива: «Я не хочу маленьких компаний; вы должны располагаться так, чтобы не оказаться с той же подругой в вашем ряду до тех пор, пока вы не проведете этих прогулок со всеми остальными». На это наши девушки заявили, что поступать так очень трудно. Поэтому мисс Фармер решила сама организовать ряды. Для начала она взяла класс с 9 ученицами. Ей удалось быстро организовать ряды. У каждой ученицы было 8 подружек, и она должна была находиться с двумя новыми подружками каждую неделю, так что мисс Фармер предусмотрела цикл в 4 недели.

Затем мисс Фармер был поручен класс с 15 ученицами. Поэтому было необходимо ввести цикл в 7 недель для того, чтобы каждая ученица была за это время в одном

ряду с 14 остальными. Тогда мисс Фармер поняла, в какое ужасное предприятие она оказалась вовлеченной.

Хотите ей помочь? Но заметьте: это вы сами пускаетесь в это ужасное предприятие. Всячески желаю, чтобы вашему микрокомпьютеру было больше нечего делать. Попытайтесь, если все предыдущее не будет отнимать долгие часы...

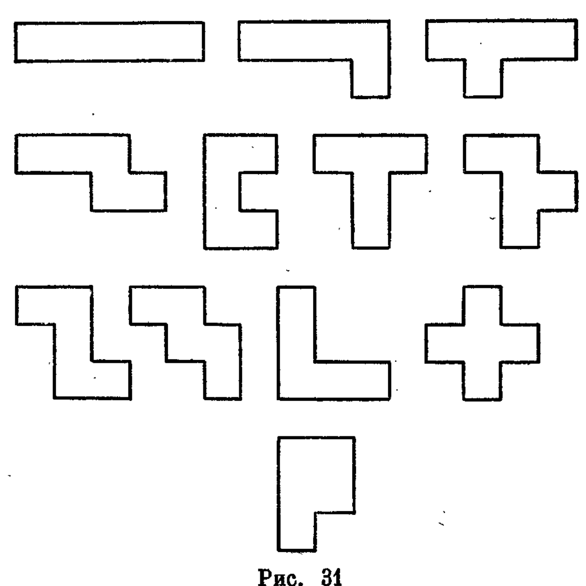


Рис. 31

Эта задача принадлежит Т. П. Киркману и была впервые опубликована в Lady's and Gentleman's Diary за 1850 год. Для ее решения нужно пролить немало чернил, и вы можете рассмотреть значения, отличные от 9 и 15. Но выглядит вполне правдоподобным, что 15 — патологическое число, и Роуз Болл [BAL] предложил геометрический подход к решению, если число девочек не равно 15. Может быть, подход с точки зрения информатики позволит вам получить в этой задаче новые результаты...

*** Головоломка 26. Пентамино.

Домино, пентамино: очевидная игра слов, заставляющая перейти от шашек с двумя квадратами к шашкам с 5 квадратами. Вот двенадцать возможных шашек, получаемых объединением 5 квадратов равной площади.

Все они приведены на рис. 31. Эти двенадцать шашек, каждая из которых имеет площадь 5, могут быть собраны

в прямоугольник с площадью 60. Есть много решений для прямоугольника 6 × 10, а также 5 × 12. Меньше решений для прямоугольника 4 × 15 и только два для прямоугольника 3 × 20, если, конечно, не различать решения, отличающиеся только симметрией. Можете ли вы найти их за разумное время на вашем компьютере, проверив, что их действительно именно 2?

Есть много путей подхода к этой задаче, даже если все они действуют согласно одной и той же стратегии: перебрать все возможные попытки. Но есть ходы, которые ни к чему не могут привести. Вы не можете начать с того, чтобы поставить Т в левый нижний угол, как на рис. 32. Ни одна шашка не может замкнуть две заштрихованные пустые клетки рядом с Т... Здесь есть необходимость и для хорошего представления позиций, но также и немного — для хитрости...

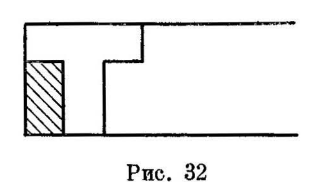


Рис. 32

* Головоломка 27. Песенка почти спета.

Знаменитая игра Армана Жаммо уже была упомянута выше (игра 12). Но сейчас мы еще не описываем ее полностью; она довольно трудна для программирования. Вот другая форма, которую проще реализовать и которая еще не лишена интереса. Я верю также, что для любителей математических развлечений здесь есть что делать.

Возьмем случайным образом p двузначных чисел. Возьмем случайным образом также двузначное число s. Соединим эти p чисел между собой сложениями или вычитаниями. Все числа должны быть использованы. Можно ли таким образом получить число s?

При последовательных испытаниях компьютер будет работать быстро. Тогда вы можете попытаться увидеть, что происходит, когда мы заставляем меняться p. Если у вас мало чисел, то у вас и мало шансов получить s. Если вы берете много чисел (большое p), то, поскольку вы обязаны использовать их все, то у вас снова мало шансов прийти к цели. Мне кажется, что наиболее благоприятны значения p около 8 или 9. Но я не осмеливаюсь гарантировать этого полностью. Нужно быть уверенным в генераторе случайных чисел. Получаете ли вы тот же результат? Я не пытался получить его математическим рассуждением. Может быть, я и неправ. Если я действительно неправ, дайте мне знать об этом!

*** Головоломка 28. Песенка спета.

На этот раз дело идет именно об игре Армана Жаммо. Вам надлежит задать вашему компьютеру шесть шашек, взятых среди 24; а именно, в набор входят:

по два раза — шашки от 1 до 10,
один раз — шашки 25, 50, 75, 100.

Затем вы задаете искомое число, скажем n, обязательно трехзначное. Требуется соединить значения шашек между собой с помощью четырех операций: сложения, вычитания, умножения и деления — чтобы получить число n. Не обязательно использовать все 6 шашек.

Если число n получить нельзя, то телевизионная игра допускает и числа, близкие к n, и тот, чье число ближе всего к n, и становится победителем.

Теоретически эта программа не должна быть трудной. Есть ограниченное число возможных комбинаций:

— есть 15 способов взять две шашки среди 6 и, самое большее, 4 способа соединить их между собой, следовательно, самое большое 60 комбинаций с двумя шашками. Но их уже гораздо больше для трех шашек. Испытать все комбинации за разумное время не представляется возможным.

Когда вы излагаете решение, вы берете две шашки из 6, соединяете их между собой одной из четырех операций (на самом деле можно считать, что только тремя. Начинать с деления — это исключение). Есть 60 (или, скорее, 45) способов это сделать. После этого задача сводится к задаче с 5 шашками. При таком подходе решение кажется более достижимым.

Следовательно, попробуем. Самые большие упрощения возникают, если вы не ищете для данного числа приближенных значений. Компьютер выводит результат, если он его находит; в противном случае он сообщает, что он решения не нашел. Вы сами можете систематически проводить одну попытку за другой. Пусть p_i, p_j, p_k обозначают три из 6 шашек. Вы можете искать решение в виде

p_i * комбинация из 5 оставшихся шашек $= n$,

$p_j + p_i$ * комбинация из 4 оставшихся шашек $= n$,

$-p_j + p_i$ * комбинация из 4 оставшихся шашек $= n$,

$\pm(p_j \circ p_k) + p_i$ * комбинация из 3 оставшихся шашек $= n$,

где $\circ$ означает одну из четырех разрешенных операций. Удивительным образом все это очень быстро и очень

часто приводит к точному ответу. Никто не запрещает вам попробовать что-то лучшее...

В соответствии с заглавием примера попытайтесь поэтому для 6 шашек 10, 10, 25, 50, 75, 100 найти 370, 369, 368...

7. ОБО ВСЕМ ПОНЕМНОГУ

В этом разделе я объединил различные задачи, среди которых далеко не все являются головоломками, по той причине, что опыт показывает: средний программист в них достигает цели не без труда. Для некоторых из них в различных книгах можно найти многочисленные решения, не всегда правильные, или — во всяком случае — не всегда хорошие, или слишком плохо объясненные. Условия этих задач могут показаться мало привлекательными. Но если в программировании вы любите именно трудности, не поддавайтесь первому впечатлению.

* Головоломка 29. Дихотомический поиск.

Это — совершенно известная задача. Вам предлагается упорядоченная таблица попарно различных элементов; например, в порядке возрастания. Вам предлагается, кроме того, другой элемент: его нужно разместить в таблицу.

Следовало бы уточнить (хоть это и не в моих правилах: обычно я предоставляю вам заботу об уточнении. В этой книге вовсе не я тот человек, который должен аккуратно работать...). Пусть a — таблица с n элементами, упорядоченная так, что

$$a[i] < a[i+1] \text{ для } 1 \leqslant i < n,$$

и x — элемент, который нужно разместить. Его место

$$\begin{aligned} &0, \text{ если } x \leqslant a[1],\\ &i, \text{ если } a[i] < x \leqslant a[i+1],\\ &n, \text{ если } a[n] < x. \end{aligned}$$

Один сотрудник факультета Нотр-Дам де ла Пэ в Намюре изучил 18 программ, опубликованных различными авторами по всему свету и в каждой нашел хоть что-то, за что можно упрекнуть. Всякий раз, когда я получаю новую книгу по программированию (к счастью, я получаю не все), я смотрю, нет ли там случайно исследования этой задачи. Почти во всех случаях это так. Настоящий «ослиный мост» *) информатики...

*) «Ослиным мостом», дальше которого учащегося сдвинуть трудно, считалась в XII—XIII вв. в Парижском университете либо

* Головоломка 30. Равенство «с точностью до пробелов».

Пусть даны две буквенные цепочки: a и b. Составьте программу, которая может сказать, совпадают ли эти цепочки с точностью до пробелов. Внимание: вы не имеете права изменять цепочки a и b, вы не имеете права порождать новые цепочки. Это запрещает вам удалить пробелы из обеих цепочек или копировать их, удаляя пробелы. Под равенством с точностью до пробелов нужно понимать, что обе цепочки должны быть образованы одними и теми же буквами в одном и том же порядке, если не учитывать пробелы. Такая задача встречается в системах, связанных с практической работой, с программами, потому что пробелы чаще всего рассматриваются в операторах и командах как незначащие.

Если вы находите это совершенно элементарным, вы можете изучить, являются ли данные цепочки обращениями друг друга с точностью до пробелов. Вы можете также увидеть, является ли цепочка палиндромом (т. е. совпадает со своим обращением) с точностью до пробелов. Так, полиндромами являются

А РОЗА УПАЛА НА ЛАПУ АЗОРА
АРГЕНТИНА МАНИТ НЕГРА

Попытайтесь получить правильную (это уж как минимум) и элегантную программу.

Головоломка 31. Анаграмма.

Еще одна головоломка, вопреки ее внешнему виду. Дело в том, чтобы сказать, являются ли две цепочки букв анаграммами друг друга (т. е. получаются ли они друг из друга перестановками букв). Эта задача имеет совершенно различный вид в зависимости от того, разрешите ли вы себе изменять обе цепочки или порождать новые цепочки, или нет. Выбор я предоставляю вам... Может быть, вы заметите, что различные решения следует оценивать в зависимости от соотношения между размерами цепочек и используемого алфавита. Подумайте о крайних случаях: алфавит из 26 букв и цепочка из 1000 символов; алфавит из 1000 символов (это вроде китайского...) и цепочка из 10 символов.

Головоломка 32. Анаграмма с точностью до пробелов.

теорема о равенстве углов при основании равнобедренного треугольника, либо геометрическое доказательство теоремы Пифагора.— *Примеч. пер.*

Та же головоломка, но, кроме того, пробелы не считаются. Вы можете ее еще немного обобщить: являются ли две страницы текста анаграммами одна другой, не считая знаков препинания?

??* Головоломка 33. Переставить две части вектора.

Вам дана таблица a с n элементами. Требуется переставить части с номерами от 1 до m и от $m+1$ до n (рис. 33).

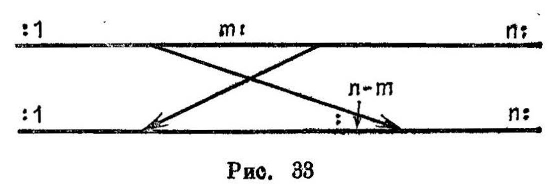


Рис. 33

Порядок элементов в каждой из частей должен быть сохранен *). Вы не должны использовать вспомогательную таблицу. Каждый элемент должен быть перемещен не более одного раза.

Это — довольно любопытная задача, которая была предложена мне Давидом Грисом, и которую он исследовал в своей книге [GRI]. Это — один из редких случаев, когда я не смог вывести программу из гипотезы рекуррентности, как я это обычно делал [ARS]. В данном случае я сначала придумал программу (ничего особенного, вы ее, конечно, прекрасно составите). И только после того — именно после того — я смог показать, почему эта программа работает правильно.

* Головоломка 34. Задача о равнинах.

Вам дается упорядоченная таблица каких-то элементов, например, телефонный справочник (где фамилии расположены в алфавитном порядке. Здесь мы не учитываем имен). В таблице могут встретиться омонимы (иначе говоря, последовательности из совпадающих элементов), как в телефонном справочнике. Требуется найти наиболее длинные омонимы: больше ли МАРТЫНОВых, чем СЕМЕНОВых?

Я использовал для этой головоломки название, данное ей в книге Давида Гриса [GRI]. Если вместо того, чтобы взять для иллюстрации таблицу фамилий, вы берете

*) Вот другая и, на мой взгляд, более правильная формулировка этой задачи: циклически сдвинуть элементы n-вектора на m позиций влево.— *Примеч. ред.*

таблицу чисел, расположенных в неубывающем порядке, то такая таблица составлена из участков возрастания, подъемов и ровных участков, «равнин». Тогда нужно найти наиболее длинную равнину.

Эта задача оказывается не вполне одной и той же в зависимости от того, ищете ли вы только наибольшую длину равнины (что делает Д. Грис) или ищете одновременно и длину ряда омонимов и сам наиболее часто встречающийся омоним (что предлагаю вам я).

С этой задачей связана неприятная для меня история. Я намеревался продумать эту задачу в Нанси также, впрочем, как и Давид Грис. Я довольно легко обнаружил два решения, различные по духу, но не по виду, что поставило передо мной задачи преобразования программ (каким образом различные отправные точки могут привести, с точностью до нескольких манипуляций, к одной и той же программе). Как и рассказывает в своей книге Давид Грис, я очень гордился своими решениями, пока не обнаружил в той же книге Д. Гриса решение, принадлежащее Майклу Гриффиту: его решение намного проще...

Сумеете ли вы найти простое решение?

??** Головоломка 35. Самая длинная возрастающая подпоследовательность.

Пусть дана таблица a из n каких-либо чисел (но если это может доставить вам удовольствие — из натуральных чисел. Это неважно). Подпоследовательность этой таблицы есть последовательность чисел, выделенная в порядке возрастания номеров. Более точно, последовательность

$$a[i_1] \;\; a[i_2] \;\; a[i_3] \ldots a[i_p]$$

есть подпоследовательность последовательности a, если $i_1 < i_2 < \ldots < i_p$. (Числа идут в одном и том же порядке в таблице a и в ее подпоследовательности, но эта формулировка двусмысленна.)

Последовательность возрастает *), если, кроме того,

$$a[i_1] \leqslant a[i_2] \leqslant a[i_3] \leqslant \ldots \leqslant a[i_p].$$

Требуется выделить из a самую длинную возрастающую подпоследовательность. Вы имеете право использовать вспомогательные векторы.

Можно найти исследование этой задачи в нескольких книгах и на нее изведено немало чернил (да и мела тоже:

---

*) Нужно было бы сказать «не убывает», но получилось бы совершенно не в стиле этой книги. — *Примеч. ред.*

я видел ее исследования в трудах международных семинаров). Кроме того, совершенно не одно и то же — довольствуемся ли мы определением максимальной длины и даже последнего члена самой длинной возрастающей подпоследовательности последовательности a (внимание: может случиться, что есть много таких подпоследовательностей одинаковой длины) или же мы хотим получить также список членов такой максимальной последовательности.

Иногда в условие вводят дополнительное ограничение: число требуемых операций должно быть порядка $n * \ln(n)$. Я не уверен, что это действительное ограничение. Если вы найдете решение, то оно, скорее всего, будет обладать этим свойством.

??** Головоломка 36. Самое длинное слово.

Заглавие вводит в заблуждение... Однажды мы проводили экзамен у наших учеников в DEUG по составлению программы, которая сообщает, скрыто ли данное слово в данной фразе, иначе говоря, встречаются ли буквы данного слова в том же порядке в данной фразе. Так, в следующей фразе (взятой из «Ярмарки у скупцов» Жана Шарля):

«Je peux te donner l'adresse d'un excellent cireur de parquets: il se rend à domicile»

слово TONDEUSE скрыто (соответствующие буквы подчеркнуты), но ни слово GAZON (нет буквы G), ни слово DOMINATEUR (все буквы есть, но в неправильном порядке) не содержатся.

Но это не головоломка, это совсем просто (уж это точно...). Я спрашиваю вас о другом — найти, какое слово наибольшей длины скрыто одновременно в двух фразах. На самом деле, конечно, речь идет не о слове, а скорее о последовательности букв: какая наиболее длинная последовательность букв может быть обнаружена в одном и том же порядке в двух фразах. Если это может вам помочь, то вот другой пример из «Ярмарки у скупцов»:

«A l'occasion du 14 juillet, les hommes remplaceront les cruches dans les chambres».

Моя программа сообщает мне, что наиболее длинная последовательность букв, которая встречается одновременно в одном и том же порядке в обоих отрывках, это набор

JETEOERLARNLECREDASLSAME

Вы можете проверить, что эти буквы действительно встречаются в обеих фразах (во второй из них они подчеркнуты). Но вручную невозможно проверить, что этот набор — самый длинный из возможных. Если вы не можете доверять вашей программе, не пишите ее...

Если вы сожалеете, что я злоупотребил названием, которое напоминает совсем о другом, а именно, об игре Армана Жаммо: найти наиболее длинное слово, которое можно образовать из 9 данных букв, то я не запрещаю вам исследовать также и эту игру. Но тут я вижу два препятствия. Во-первых, с точки зрения процесса ее создания есть очень мало того, что требуется обнаружить или открыть (если не пришлось открывать какой-нибудь словарь Скраббля). Далее, нужно ввести в компьютер настоящий словарь, что предполагает большой объем хранения и чудовищный труд отстукивания по клавишам, совершенно лишенный интереса...

?*** Головоломка 37. Белый прямоугольник.

Ах, ах! Тут-то я вас и поймал. Вы немедленно вообразили себе какую-нибудь не поддающуюся пересказу историю... Такое в книгах по информатике бывает редко, но бывает [SIK]. В своем сочинении о языке ЛИСП Лоран Сиклосси должен был использовать многочисленные примеры буквенных цепочек, и все составленные им цепочки одна смешнее другой. Это и вдохновило меня на предыдущую головоломку. Я предложил своему издателю перевести сочинение Сиклосси, что не должно было быть таким уж трудным, поскольку автор использует французские фразы, чтобы блеснуть учтивостью (иначе он бы говорил это по-латыни). Но Сиклосси, который превосходно владеет французским языком — несмотря на то, что он профессор информатики в Соединенных Штатах,— захотел изменить примеры к французской версии книги, используя «акрофонические перестановки» — перестановки букв или слогов, создающие слова с новым значением... Издатель не согласился, и французская литература потеряла прекрасное сочинение о языке ЛИСП... Если вы читаете по-английски и если вы хотите выучить ЛИСП, почему вам нельзя развлекаться, учась?

Здесь речь идет совсем о другом. Эта задача была предложена как одна из четырех тем на конкурсе программирования в АФСЕТ несколько лет назад. Вам дана прямоугольная решетка для кроссворда. Найдите белый (т. е. не содержащий вычеркнутых черных клеток) прямоуголь-

ник наибольшей площади, вписанный в решетку (квадрат есть частный случай прямоугольника).

На рис. 34 есть прямоугольник площади 8 в левом нижнем углу и есть квадрат площади 9. Это — хороший ответ. Программа, которую вы должны составить, должна читать размеры сетки (число строк и столбцов), затем — координаты черных полей и, наконец, давать прямоугольник наибольшей площади, например, указанием координат двух противоположных вершин.

Для программистов на конкурсе АФСЕТ это не было легкой задачей. Она оказалась едва ли не наиболее трудной задачей, будучи единственной задачей, доставившей мне затруднения (см. головоломку 22, другое упражнение на том же конкурсе). Один из соревнующихся достиг цели, решительно пренебрегая эффективностью. Вы-то не очень ограничены временем (по крайней мере временем размышления или временем программирования). Попытайтесь составить программу, время вычисления которой не слишком быстро растет вместе с размером сетки.

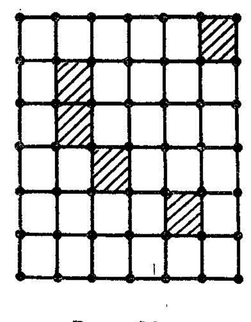


Рис. 34

Головоломка 38. Математическая композиция.

Ж.-К. Байиф [BAI], французский язык которого очень отточен, представил эту головоломку под названием «арифметическая композиция» в своей книге, из которой я ее и заимствую. Композиция состоит из четырех вопросов, связанных с вычислением площади. Один из вопросов относится к полной поверхности куба, сторона которого измеряется целым числом метров. Вот ответы учеников на различные вопросы:

Альбер	8	16	12	16
Бернар	12	16	12	18
Клод	12	18	12	18
Дени	16	18	12	20
Эрнест	8	16	10	16
Фернан	12	12	12	22
Гюстав	16	18	12	20
Анри	8	16	10	16
Исидор	16	12	12	20
Жюль	20	12	12	20

(Это — величины площадей в квадратных метрах, предложенные учениками.) Преподаватель ставит 5 за верный ответ и 0 за неверный ответ. Один-единственный из учеников получил 0. Кто оказался первым? Вне всякого сомнения, вы можете сказать больше. Эта головоломка кажется мне особенно подходящей для ее решения на компьютере...

?? Головоломка 39. Другая головоломка Давида Гриса.

Пусть дан вектор, образованный n целыми числами. Подпоследовательностью этого вектора называется набор элементов, в котором индексы идут подряд. Найти подпоследовательность с максимальной суммой. Если вы предпочитаете другую формулировку, то найдите индексы i, j, для которых величина

$$a_i + a_{i+1} + \ldots + a_{j-1} + a_j$$

максимальна. Внимание: время вычисления не должно расти намного быстрее, чем n, когда n увеличивается...

Эта головоломка до некоторой степени напоминает головоломку о возрастающих подпоследовательностях, но она гораздо менее сложна. Подумайте: линейная зависимость от n! Да ведь это, грубо говоря, означает, что каждый элемент вектора рассматривается только один раз. Вам следует составить цикл

ДЛЯ i := 1 ШАГ 1 ДО n ВЫПОЛНЯТЬ...ВЕРНУТЬСЯ

И никаких циклов в этом цикле! В ответе вы получаете искомые индексы. Озадачивающе, не так ли? Я потерял на это немало времени. И тем не менее, какое простое решение!

Как вы находите?

ЧАСТЬ II
ПЕРВАЯ ПОМОЩЬ

У вас возникли затруднения,— вы не знаете, как приступить к игре или головоломке. Я не хочу предлагать вам готовое решение, иначе в чем будет удовольствие? Но, вероятно, нижеследующие указания наведут вас на правильный путь. Если бы я мог использовать дискетку вместо книги, я разложил бы эти указания на как можно большее число уровней, чтобы вы были в состоянии брать из них тот строгий минимум, который позволит вам продолжать решение. Но и в книге вы всегда можете прочесть только начало и — как только искра вспыхнет — отложить книгу и завершить решение самостоятельно.

Некоторые упражнения кажутся мне настолько очевидными, что я ничего про них не скажу. Но, конечно, это субъективная точка зрения: они могут не казаться очевидными вам. Может быть, потому, что вы не видите, в чем задача: разберите простые случаи вручную. В любом случае старайтесь сформулировать задачу с максимальной возможной полнотой. Если и после этого вы попрежнему не видите, как вам ее охвзтить, поговорите с друзьями или с приятелями. Я умышленно не собираюсь сообщать вам все. Эта книга написана для того, чтобы вы стали программировать. Вам играть...

1. СЛУЧАЙНЫЕ ЧИСЛА

Упражнение 2.

Нужно изучить поведение дробной части $(x+a)^8$, когда x меняется от 0 до 1. Нарисуйте, хотя бы приближенно, кривую, представляющую эту функцию. Рассмотрите интервал на оси x, в котором значение функции меняется от некоторого целого числа до следующего за ним. Отметьте на кривой точку, в которой ордината равна этому целому, увеличенному на 0,5. Она разбивает область изменения x на два интервала. Равны ли между собой эти две половинки? Если одна из них больше другой — и если это одна и та же половинка для

всех интервалов — то у вас больше шансов получать числа, меньшие (или большие, вам будет видно самим), чем 0,5.

Но что касается выбора a, то напомним, что следует избегать соотношения

$$\text{дробная_часть}\ ((x+a)^8) = x,$$

иначе вы вместо случайной последовательности получите постоянную последовательность. Проверьте числа $x = 0$, $x = 0{,}5$ и $x = 1$.

Упражнение 4.

Вы располагаете генератором случайных чисел, дающим число, содержащееся между 0 и 1, и вы хотите получить случайным образом число между 1 и 6, включая границы. Тогда остается сказать, что вам нужно различать 6 случаев и приписать каждому из случаев значение одного из этих целых чисел.

Почему не разделить интервал (0,1) на 6 частей?

Или еще по-другому: почему бы не умножить выше случайное число на 6. Тогда оно окажется в интервале (0, 6), исключая 6. Если вы возьмете целую часть результата, то вы получите целое число от 0 до 5, включая границы, с равными вероятностями для каждого числа... Завершить следует вам, я уже сказал слишком много!

Игра 1.

Если вы знаете, как сделать предыдущее упражнение, то это для вас уже не задача. Нужно подделать кости, иначе говоря — сделать так, чтобы одна из граней выпадала чаще остальных. Это должно означать, таким образом, что вместо того, чтобы делить интервал (0, 1) на 6 равных частей, нужно взять 5 частей равных между собой, а шестую побольше. Легко! Наиболее простое решение состоит в умножении случайного числа на целое, большее 6, и в присвоении новых значений грани, которую вы решили предпочесть.

Элементарно, мой дорогой Ватсон!

Игра 2.

Х.-К. Байи упростил задачу, указав две возможные стратегии:

— бросать кость до тех пор, пока не будет достигнута некоторая намеченная заранее сумма (по крайней мере если игрок не будет остановлен по дороге выбрасыванием единицы);

— бросать кость определенное число раз, намеченное заранее.

В первом случае предположим, что уже имеющаяся у вас сумма равна n и что вы собираетесь осуществить еще одно бросание. У вас есть один шанс из 6 получить каждое из следующих шести чисел: 0, $n+2$, $n+3$, $n+4$, $n+5$, $n+6$. Если вероятный выигрыш не увеличивает полного выигрыша (если среднее из этих чисел меньше n), то играть не следует. Вы должны получить $n=20$.

Если вы бросаете кость 6 раз, то — поскольку все грани имеют равные шансы выпасть — вы должны проиграть. Это не слишком строгое рассуждение, но короткое... Если единица вам не выпала, то у вас один шанс из пяти получить числа от 2 до 6, что дает в среднем 4. За 5 ходов получаем 20. Это — еще один способ получить оценку для числа ходов.

Но есть и другие возможные стратегии. Вы можете, в частности, решить останавливаться в зависимости от того, какое из двух событий наступает первым: сумма, большая 19, или число ходов, равное 5.

Используйте ваш компьютер, чтобы произвести соответствующие опыты.

Если вы хотите взглянуть на это с точки зрения искусственного интеллекта, то вы можете также снабдить вашу программу механизмом самообучения. Вы помещаете в вашу программу три упомянутые выше стратегии. Розыгрыш определяет случайным образом ту, которая будет использована в каждой из партий. Вначале все три стратегии имеют равные вероятности. Если выбранная стратегия выигрывает, то вероятность ее применения увеличивается. Если она проигрывает, то ее вероятность уменьшается. Чем больше вы играете, тем чаще компьютер должен выигрывать. После очень большого числа партий полученные частоты применения стратегий скажут вам, какая из них является наилучшей.

Головоломка 1.

Это — нетрудная программа, разве что вы не взяли на себя заботу четко сформулировать задачу. Последовательность целых чисел, порождаемая этой программой, является так называемой возвратной последовательностью, каждый член которой полностью определяется значением предыдущего члена:

$$u_i = f(u_{i-1}).$$

Сказать, что последовательность u_i становится периодической — то же, что сказать, что существует некоторое p, для которого

$$u_{i+p} = u_i$$

для достаточно больших i. Но если это выполняется для данного i, то

$$u_{i+p+1} = f(u_{i+p}) = f(u_i) = u_{i+1}$$

и, следовательно, $u_{j+p} = u_j$ для любого j, большего i. Пусть r — наименьший из индексов, для которых $u_{r+p} = u_r$.

От вас не требуют найти число r, нужно найти только число p. Можно предложить два решения:

— если i — достаточно большое число, кратное p, то $u_{2i} = u_i$;

— выберите исходное значение d и длину интервала h. Для любого i от $d+1$ до $d+h$ посмотрите, не равно ли соответствующее значение u числу u_d. Если равно, то вы нашли период и все закончилось. Если же никакого равенства не получается, то либо d меньше, чем r, либо h меньше p, либо и то, и другое. Попытайтесь сделать то же еще раз с бóльшими d и h.

Есть много способов реализовать вторую из этих стратегий. По крайней мере в некоторых случаях она быстрее первой.

Головоломка 2.

Совершенно ясно, что вы не можете начинать проводить какие-либо статистические подсчеты до того, как вы реализуете m бросаний. Наш маленький вундеркинд хотел бы сделать единственный цикл, в котором $m-1$ первых ходов подвергаются специальной обработке. Это — совершенно бесполезная сложность. Составьте первый цикл по данным m первым ходам. Затем — второй цикл, проводящий статистику.

Наш маленький вундеркинд совершил и вторую ошибку, для меня еще более необъяснимую: он объединил последовательные ходы в таблицу. Но это совершенно бесполезно. В любой момент единственное, в чем вы нуждаетесь, это в результатах m последних бросаний. С каждым новым бросанием результат наиболее старого из учитывающихся ранее бросаний теряет силу. Поэтому вы можете его упразднить. Если и есть таблица, то ее размер m, а не n!

Но не очевидно, каким образом хранить в таблице m бросаний. Вы можете представить их в виде m символов, образующих цепочку. На каждом ходе цепочка теряет свой последний символ и получает новый первый символ.

Но можно сделать еще и по-другому. Речь идет об «орле» и «решке». Нам нужно только два различных символа, например, 0 и 1. Эти m символов 0 и 1 могут рассматриваться как цифры числа в двоичной записи. Тогда вам не нужна ни таблица, ни цепочки символов. В соответствии с выбором нужно выполнить либо умножение на 2 (что сводится к одному сложению), либо деление на 2.

Относительные успехи трех наших решений зависят от используемой вами системы. В зависимости от управления, принятого для таблиц и цепочек, в зависимости от искусства программиста, составившего систему интерпретации вашего языка высокого уровня, либо таблица одолевает цепочки, либо наоборот.

В составленной мною системе на языке LSE использованы двоичные числа, дающие несколько лучший результат, чем полученные с помощью цепочки, которые, в свою очередь, дают заметно лучший результат, чем полученный с помощью таблиц.

Игра 3.

Единственная трудность в этой программе: перетасовать карты. Я уже упомянул об этом, описывая условия игры. Есть много возможных идей:

— приготовить сначала карточную колоду, затем вытаскивать их из стопки одну за другой. В этом случае у вас будет выбор, как поступать:

либо расположить карты в таблицу с 52 полями,

либо создать цепочку из 52 символов.

Но нужно ли это на самом деле? Почему бы не исходить из простой начальной ситуации: упорядоченной таблицы или отсортированной цепочки, затем выбирать элемент этого множества с помощью случайного бросания, вынимать его из множества и повторять процедуру с меньшим количеством элементов.

Если так поступать, то применение таблицы становится тонкой задачей: как изъять элемент из множества?

Его можно изъять «физически». Все элементы, расположенные выше него, спускаются в таблице вниз на одну ступеньку. Это сохраняет порядок оставшихся элементов.

Но нужно ли это? Почему бы, что гораздо проще, не переставить выбранный элемент с последним элементом таблицы в процессе выполнения операции?

Как только мы это обнаружили, становится очевидно, что в перетасовывании карт, исходя из начальной колоды, больше никаких трудностей нет: вы размещаете колоду в упорядоченную таблицу из *n* карт, вы выбираете случайным образом целое число между 1 и *n*, вы меняете местами соответствующий элемент с элементом *n*, затем вы уменьшаете *n* на единицу и повторяете процедуру.

Элементарно, когда все испробовано!

Игра 4.

Я уже дал все необходимые пояснения, кроме порождения лабиринта. Первую попытку я предпринял со следующим алгоритмом:

— поставить *i* в начальное положение (правый нижний угол),

— выбрать случайным образом направление перемещения (целое от 1 до 8); если это перемещение невозможно — перейти к следующему перемещению, пока не будет найдено возможное перемещение;

— передвинуть *i* в соответствии с этим перемещением;

— если *i* оказался на поле прибытия, то все закончено, в противном случае повторить процедуру.

Опыт показывает, что чаще всего эта программа не останавливается.

Так как есть основное направление перемещения, подлежащее реализации (диагональ игры), то я изменил случайный выбор так, чтобы сделать более частными перемещения влево и вверх или вверх и влево.

Так как у меня еще были и другие задачи, я решил останавливать случайный выбор, когда *i* оказывается в маленьком прямоугольнике вверху слева. Полученный реестр (сделанный из таблицы или цепочки символов) дает тогда путь, ведущий из каждой из точек этого прямоугольника к полю прибытия.

Остальное просто.

Игра 5.

Эта игра не представляет никаких трудностей. Пусть вы не пытаетесь гарантировать Тони возможность выхода. В программе — никакой стратегии: один ход на два поля, два препятствия на горизонтальной линии на двух свободных полях (вы выбираете клетку случайным образом. Если она или ее соседка справа не свободны, то вы повторяете выбор. Если они свободны — вы их помечаете. Тем

хуже для Тони, если он накрыт), два препятствия случайным образом на двух вертикальных полях.

Если вы решили оставить Тони шансы на спасение, действуйте, как в предыдущей программе. Вы случайным образом выбираете путь и обозначаете его так, чтобы никакое препятствие на него сверху не падало. Но вы не выводите на экран этих обозначений.

2. ИГРЫ С ЧИСЛАМИ

Головоломка 3.

Я нашел это упражнение в монографии, посвященной языку Пролог. Предложенное там решение действует методом проб и ошибок. Но задача решается намного проще.

Как всегда, полностью определим задачу. Искомое число представляется в десятичной системе последовательностью цифр

$$c_n c_{n-1} \ldots c_2 5$$

Умножая на 5, получаем

$$5 c_n c_{n-1} \ldots c_3 c_2$$

Отсюда следует, что $c_2 = 5$. Все цифры c_i точно так же итеративно вычисляются справа налево, обыгрывая оставшееся от предыдущего умножения «в уме»: когда вы умножаете крайнее справа 5 на 5, вы получаете 5 единиц, что и дает $c_2 = 5$, и 2 «в уме». Тогда вы можете вычислить c_3 и новую цифру «в уме» и продолжать шаг за шагом. Остается маленькая задача о том, как узнать, когда следует остановиться. Изучите ее сами; как обычно, я не хотел бы сообщать вам все...

Вы можете также действовать слева направо:

$$5 c_n c_{n-1} \ldots c_3 c_2 : 5 = c_n c_{n-1} \ldots c_3 c_2 5$$

Деля левую цифру на 5, вы получаете $c_n = 1$. Имея c_n, вы можете продолжать деление. И здесь тоже вам нужно будет принимать во внимание перенос результата, полученного при предыдущем делении, и нужно будет знать, когда остановиться. Эти два метода по существу равносильны.

Остальное оставляю исследовать вам.

Головоломка 4.

Обычно я бываю глубоко разочарован тем, что нахожу в книгах по информатике или по математике касательно квадратных корней. Чаще всего вам предлагают метод

Ньютона: пусть вам нужно извлечь квадратный корень из числа x. Вы образуете возвратную последовательность u_i по правилу

$$u_{i+1} = (u_i + (x/u_i))/2.$$

Вне всякого сомнения, вы можете взять $u_0 = 1$ в качестве начального значения. Эта последовательность очень быстро сходится к квадратному корню из x. Если, например, взять $x = 50$ и воспользоваться формулой

$$u_{i+1} = \text{целая_часть } ((u_i + (x/u_i))/2),$$

чтобы иметь дело только с целыми числами, то в качестве последовательных значений u вы получите

$$u_0 = 1,\ u_1 = 25,\ u_2 = 13,\quad u_3 = 8,\quad u_4 = 7,\quad u_5 = 7.$$

Чтобы использовать здесь этот алгоритм, вы должны написать программу целочисленного деления двух целых чисел большой длины.

Другой способ действия основан на том факте, что разность двух последовательных квадратов есть нечетное число:

$$(n + 1)^2 - n^2 = 2n + 1,$$

так что последовательные разности являются последовательными нечетными числами. Поэтому можно видеть, что сумма нечетных чисел от 1 до $2k - 1$ включительно есть k^2. Обратно, если вычитать из n последовательно возрастающие числа, пока это возможно (не допуская, чтобы результат становился отрицательным), тогда искомый квадратный корень есть k, если последнее нечетное вычитаемое равно $2k - 1$. Таким образом, для 50

$$\begin{aligned}
50 - 1 &= 49,\\
49 - 3 &= 46,\\
46 - 5 &= 41,\\
41 - 7 &= 34,\\
34 - 9 &= 25,\\
25 - 11 &= 14,\\
14 - 13 &= 1.
\end{aligned}$$

Нельзя продолжать, не получая отрицательной разности. Последнее нечетное вычитаемое равно 13, поэтому корень есть $(13 + 1)/2 = 7$ (и остаток 1). Этот способ гораздо лучше подходит для распространения на случай очень больших чисел, потому что вам требуется реализовать только две операции:

— прибавить 2 к большому числу;
— вычесть одно большое число из другого.

Но число шагов цикла равно искомому квадратному корню, а он может оказаться весьма большим.

Можно обобщить предыдущий алгоритм, используя свойства десятичной записи чисел. Данное число разделяется на куски по две цифры, начиная справа; затем мы начинаем вычитать последовательные нечетные числа из крайнего слева куска:

$$\begin{array}{r} 1909 \\ -1 \\ \hline 1809 \end{array} \quad \begin{array}{r} 1809 \\ -3 \\ \hline 1509 \end{array} \quad \begin{array}{r} 1509 \\ -5 \\ \hline 1009 \end{array} \quad \begin{array}{r} 1009 \\ -7 \\ \hline 309 \end{array}$$

Если это нельзя продолжать дальше, то последнее вычитаемое число увеличивается на единицу, сдвигается на один шаг вправо, и следом за ней приписывается единица. Это — первое нечетное число, которое следует вычитать из предыдущего остатка.

В приведенном выше примере $7 + 1 = 8$; приписывая 1, получаем 81 и продолжаем:

$$\begin{array}{r} 309 \\ -81 \\ \hline 228 \end{array} \quad \begin{array}{r} 228 \\ -83 \\ \hline 145 \end{array} \quad \begin{array}{r} 145 \\ -85 \\ \hline 60 \end{array}$$

Поскольку продолжать дальше нельзя (последнее возможное вычитание из остатка — это крайнее справа), то последнее из вычитаемых чисел нужно увеличить на 1, а затем разделить на 2, чтобы получить корень. Последний остаток и есть остаток квадратного корня:

$$85 + 1 = 86, \quad 86/2 = 43,$$
$$1909 = (43)^2 + 60.$$

Этот алгоритм достаточно прост для программирования при длинных числах, и он дает вполне разумное время вычисления.

У вас много возможностей представлять свои данные. Так как мы оперируем с кусками из двух цифр, то вы можете задавать свои данные таблицами целых чисел в интервале от 0 до 99.

Вы можете представлять свои целые числа как цепочки символов, где используются только числовые символы (цифры) от 0 до 9. Выбор способа зависит от ваших предпочтений и от возможностей вашей машины оперировать с таблицами и цепочками. Тщательно рассмотрите, какие операции нужно сделать. Вы ничем не ограничены: почему бы не запрограммировать и сравнить два разных решения?

Я предложил вам алгоритм без доказательства. Поэтому попытайтесь его проверить...

Я предложил вам алгоритм для десятичной системы счисления. Можно предложить похожий алгоритм для двоичной системы. Тогда не возникнет цикл вычитаний последовательных нечетных чисел из каждого куска, поскольку в куске есть только одно нечетное число: 1. Алгоритм упрощается: если можно вычесть нечетное число — мы его вычитаем, в противном случае мы не делаем ничего. Затем сдвигаем, добавляем 1 и приписываем 1 в конце... Этот алгоритм намного легче реализовать. Но тогда нужно сначала перейти к основанию 2, а затем преобразовать двоичный результат в десятичный. Вам следует посмотреть, что более эффективно...

Головоломка 5.

Аккуратно поставим задачу. То, что от вас требуется, — это не взятая глобально последовательность, а вот что: если начало последовательности выписано, то нужно найти следующее число. Возьмем пример, данный в головоломке 5: какое число следует за 50?

Есть ровно три возможности.

1. Число делится на 2. После однократного деления на 2 оно не будет иметь других делителей нуля, кроме 2, 3 и 5. Следовательно, это число — из последовательности. Так как $50 : 2 = 25$, то полученное частное больше, чем 25. Наименьшее число последовательности, большее 25, есть 27. Таким образом, если следующее за 50 число делится на два, то оно равно $2 \times 27 = 54$.

2. Оно делится на 3. То же рассуждение. $50 : 3 = 16{,}7$. Первое число последовательности, большее 16,7, есть 18. Если следующее за 50 число делится на 3, то это число равно $3 \times 18 = 54$.

3. Оно делится на 5. $50 : 5 = 10$. Следующее за 10 равно 12,

$$5 \times 12 = 60.$$

Таким образом, у нас 3 кандидата: 54, 54, 60. Наименьшее из этих трех и есть искомое.

Мы получили 54, используя только уже вычисленную часть последовательности Хэмминга.

Я предложил вам идею решения на примере. Вам следует ее обобщить, показать, что это всегда верно, и составить хорошую программу для решения.

Головоломка 6.

Я предлагаю вам начать с образования различных числовых последовательностей, получаемых вычеркиванием чисел. Вот первые из них:

1 : 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19

2 : 3 5 7 ~~9~~ 11 13 ~~15~~ 17 19 ~~21~~ 23 25 ~~27~~ 29 31 ~~33~~ 35

3 : 5 7 11 13 17 ~~19~~ 23 25 29 31 ~~35~~ 37 41 43 47 ~~49~~

На этом уровне можно поверить, что появляется возвратное соотношение: во второй последовательности нет четных чисел, в третьей — нет кратных трем. Образуем следующую: 25, кратное 5 содержится. Покажем механизм перехода от одной последовательности к другой последовательности

3 : 5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49
5 : 7 11 13 17 23 25 29 31 37 41 43 47

Если вы все это хорошо поняли, то вы теперь должны суметь обобщить. Обозначим через $g\ (i,\ j)$ число, стоящее в последовательности ранга i, которая начинается с $g\ (i,\ 0)$. Число $g\ (i,\ 0) = h\ (i)$ и есть счастливое число ранга i. Если вы можете построить $g\ (i + 1,\ j)$, исходя из $g\ (i,\ \ldots)$, то вы должны суметь решить задачу. Само собою разумеется, что таблица чисел g не должна участвовать в программе. Это — только промежуточное средство вычисления...

Головоломка 7.

Нужно попытаться сгруппировать эффект нескольких последовательных шагов. Нечетное p дает $(3p + 1)/2$, которое можно еще переписать в виде

$$3\ (p + 1)/2 - 1,$$

что дает правило:

добавить 1,
разделить на 2 и умножить на 3,
уменьшить на 1.

Предположим, что результат нечетен. За операцией «уменьшить на 1» сразу же следует операция «добавить 1», и в результате этих двух операций ничто не меняется. Отсюда следует новое правило:

добавить 1,
пока результат четен, делить его на 2 и умножать его на 3,

уменьшить на 1,
делить на 2, пока это возможно.

Составьте по этому правилу программу и заставьте ее перечислять все величины, полученные таким образом (все они будут нечетны. Заметьте, что только первое число в ряду может оказаться кратным трем).

Если вы замените 3 на m, то второе правило изменяется: пока результат четен, делить его на 2 и умножать его на m.

Вернемся к случаю числа 3. Наше правило можно переписать следующим образом: пусть k — некоторое нечетное число; тогда $2^p k - 1$ дает $(3^p k - 1)/2^q$.

Назовем эту операцию переходом p, q.

Можете ли вы показать, что:

если $n = 2$ по модулю 3, то элемент, следующий за n, равен некоторому элементу, следующему за $(2n - 1)/3$;

если n дает некоторое n при переходе p, q, где $q > 1$, то число $(n - 1)/2$ порождает ту же последовательность, что и n, за исключением, быть может, нескольких первых членов.

Любое число вида $n = 4k + 1$ имеет непосредственно следующее $n' < n$.

Для того чтобы n допускало переход p, 1, необходимо и достаточно, чтобы n имело вид $n = k2^p - 1$, где

$k = 1$ по модулю 4, если p нечетно,
$k = 3$ по модулю 4, если p четно.

Если вы хотите проверить с помощью программы, что это свойство выполняется для любого нечетного n в данном интервале от 3 до n, вы можете пробежать все нечетные числа в возрастающем порядке и проверить, что для каждого из них это верно. Но вы можете сначала вычеркнуть из списка все нечетные числа, о которых вы знаете, что их поведение сводится к поведению последовательности, относящейся к меньшему нечетному числу, поскольку список нечетных чисел пробегается в возрастающем порядке, и этот случай уже был изучен. Таким образом, остается не больше чисел, чем уже было отмечено.

Но построить список априори, без вычеркиваний в более широком списке, так же трудно, как построить последовательность счастливых чисел...

Затем можно пытаться сделать еще один шаг: для любого не вычеркнутого n вычислить первый следующий за ним элемент. Он больше n (в противном случае n был бы вычеркнут). Если он содержится в интервале от 3 до N,

то мы ничего не делаем (этот случай будет изучен ниже). Если же он больше N, то мы помещаем его в резерв. Таким образом, мы получим некоторый список чисел, больших N. Если для каждого числа из этого списка возвратная последовательность достигает 1, то мы сможем доказать, что это свойство выполняется для всех чисел, меньших N, и еще для некоторых других.

Конечно, это не доказывает общей теоремы: для любого n предложенная последовательность достигает 1. Но нужно присоединить к делу новую форму рассуждения, которая потребует серьезных размышлений и надежных логических оснований для того, чтобы оказалось возможным поправить дело...

Вот, наконец, последнее свойство, которое вы должны уметь доказывать: не существует пар p, q, где p и q — натуральные числа, для которых n дает n при переходе p, q. Это не означает, что не существует периодических последовательностей. Про них я сумел доказать только тот факт, что не может иметь места цикл

n дает n' при переходе p, q;
n' дает n при переходе p', q'.

Как бы то ни было, этого на сей раз недостаточно.

Но это полезно, чтобы увидеть, каким образом 3 играет существенную роль в этом деле...

Зашифрованные операции

Общая идея состоит в том, чтобы перепробовать все возможные комбинации, согласующиеся с условием, и сохранить только те из них, которые удовлетворяют предложенной операции.

Головоломка 8.

Пусть даны значения D и E (значения различны). Из них получается Y и то, что «в уме». По этой величине «в уме» получается значение N. Так как $N + R +$ «в уме» $=$ $= E$ (плюс, быть может, 10) и так как E известно, то только N можно выбирать произвольно. Кроме того, нужно, чтобы оно отличалось от D, E и Y и нужно, чтобы R, полученное таким образом, отличалось от D, E, Y, N. Если пока все идет хорошо, то вы продолжаете выбор. Если уже возникла невозможность, то вернитесь назад и осуществите другой выбор N. Если никакой выбор для N не оказывается возможным, вернитесь назад и измените выбор E . . .

Это — одно решение.

Но оно может потребовать много времени. Чтобы выиграть время, ограничьте возможные выборы. Очевидно, что значение SEND ограничено числом 9999, как и MORE, и поэтому значение MONEY не может превосходить 19998. Так как это — число из пяти цифр, то M = 1. Это освобождает вас от испытания 1 для D и E. Если цифра единиц суммы $D + E$ равна 1, то этот набор D и E недопустим.

Поставьте 1 на свое место:

```
 SEND
+1ORE
1 ONEY
```

$S + 1 +$ «то, что в уме» дает число, большее девяти Это возможно только в случае, если мы предположим что «в уме» для S кое-что есть:

S + 2 = 10 + O

(справа буква O, а не цифра ноль).

S + 2 может превосходить 9 только в случае, если S больше 7. Единственные возможные значения — это

S = 8, что дает букве O значение 0,
S = 9, что дает букве O значение 1.

Но 1 уже присвоено букве M. Следовательно, S = 8 и O = 0.

Метод, использованный в этом упражнении, имеет очень широкую область применения. Нужно исследовать все возможности, чтобы выявить те, которые удовлетворяют условию задачи. Мы упорядочиваем их таким образом, чтобы, переходя от одной комбинации к следующей, пересмотреть их все и притом по одному разу.

1. Берем первую комбинацию.

2. Испытываем ее. Если она удовлетворяет требованиям, запоминаем ее значение.

3. Если это — последняя комбинация, то все значения записаны и все кончено.

4. Если не последняя, то переходим к следующей комбинации и повторяем, начиная с пункта 2.

В данном случае — так как мы уже знаем значения букв S, O, M, остается только три еще не определенных значения: D, E, N. Для каждой из них берем постепенно возрастающие значения, изменяя их таким образом, чтобы сначала возрастало N при постоянных D и E. Затем

меняется E при постоянном D (а N пробегает все возможные значения). Когда все возможные значения для E испытаны, мы переходим к следующему значению D.

Таким образом, D может принимать 7 значений.

Для каждого из них E может принимать 6 значений.

Для каждой такой пары N может принимать 5 значений.

Отсюда следует, что нужно перепробовать $7 \times 6 \times 5 = 210$ значений, что совершенно не затруднит компьютер...

Головоломка 9.

Будем действовать, как в предыдущей задаче. Но здесь есть некоторая дополнительная информация. В условии участвуют 10 букв:

H E L P T Y O U N G

Так как они имеют значения в виде 10 цифр, где каждая цифра участвует и притом только один раз, то

$$H + E + L + P + T + Y + O + U + N + G = \\ = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45.$$

Если вы учтете очевидные значения букв Y, O, H, то вы сможете дать сначала значения каждому из чисел «в уме». Используя тогда соотношения между значениями букв, заданных в зашифрованном сложении, вы сможете получить соотношение между четырьмя буквами и вывести из него, что E нечетно. Отсюда вы быстро выведете, что оно может принимать не более двух значений: 3 и 5.

Испытайте их одно за другим...

Головоломка 10.

Здесь снова используются 10 цифр. Вы знаете их сумму. Она делится на 9. Вы знаете кое-что о сумме цифр результата.

Вы легко сможете заменить это умножение сложением. В нем вы сможете определить все величины «в уме». Вам останется сделать не так уж много попыток...

Головоломка 11.

Эта головоломка намного серьезнее. Если вы пойдете по пути систематических испытаний, то вы рискуете потерять время зря. Есть $9! = 362880$ перестановок девяти первых цифр. Не все они подлежат проверке, поскольку крайняя слева цифра не может превосходить 3. Но остается еще очень много возможностей.

Запишите это символическое умножение и обозначьте его величины «в уме». После умножения на 3 величина «в

уме» может быть только 0, 1 или 2. Замечая, что все 9 цифр, отличных от 0, использованы, вы можете узнать сумму величин «в уме» (10). Так как 6 не может быть связано с 2 «в уме», поскольку $3 \times 6 + 2 = 20$ дает 0 в качестве цифры единиц, а это исключено, то вы сможете таким образом полностью определить величины «в уме», связанные с этими двумя цифрами. Это разрешает задачу о решениях, оканчивающихся на 3.

Так как величины «в уме» являются ключом к задаче, составьте маленькую таблицу, показывающую для каждой цифры, как она может быть получена в качестве цифры единиц произведения некоторой цифры на 3 с добавлением величины «в уме». Например, 5 можно получить как $3 \times 5 + 0$, $3 \times 8 + 1$, $3 \times 1 + 2$.

Если число кончается на 9, то результат кончается на 7, и 2 оказывается «в уме». Можно почти закончить вручную. Во всяком случае вручную легко найти какое-то решение. Программа для компьютера остается необходимой для того, чтобы найти все остальные решения.

Головоломка 12.

Легко! Чтобы доказать эту теорему, достаточно доказать, что ее утверждение справедливо для любого n, кратного трем. Давайте-ка их все переберем. Сначала для каждого n вычислим первое число n, сумму кубов цифр числа n. Если n' меньше n, то дальше идти незачем. Покажите, что n' кратно трем. Если оно меньше n, то оно уже испытано, и для него результат известен.

Можете ли вы найти такое k, что при $n > k$ имеем $n' < n$?

Если можете, то достаточно проверить искомое свойство для всех n, кратных трем и меньших k. Это делается очень быстро.

Головоломка 13.

Эти варианты исследуются таким же способом. Проделайте сначала вручную пробы, чтобы увидеть, как ведут себя последовательности сумм кубов цифр для чисел n, не кратных трем, различая случаи: n на единицу больше кратного трем, n на 2 единицы больше кратного трем.

В случае суммы квадратов вы знаете, какой результат нужно доказывать. Это легко...

Головоломка 14.

Изучаемое число имеет вид $1000a + 100b + 10c + d$ при $a \geqslant b \geqslant c \geqslant d$. И, так как не все цифры одинаковы, то непременно $a > d$.

Вы можете доказать, что результат первого вычитания

кратен девяти, так что, переходя к первой разности, вы кое-что знаете о сумме $a + b + c + d$.

Каково бы ни было исходное число, первая из полученных разностей имеет вид $999u + 90v$ с $v < u$, $0 \leqslant v$, $0 < u \leqslant 9$. Так что не так уж много чисел нужно испытывать...

Головоломка 15.

Эта головоломка намного труднее. Используйте все данные задачи, хотя и кажется, что их не слишком много.

Господин P не может найти искомые числа. Следовательно, число p не является произведением двух простых чисел — в противном случае их разложение на множители было бы однозначным.

Господин S это знает. Но их сумма s может быть многими способами представлена в виде суммы двух чисел. Ни одна из этих пар не является парой простых чисел. Это условие гораздо более ограничительно: нужно вычеркнуть из списка возможных значений s все такие значения, которые являются суммами двух простых чисел — таковы 12 (так как $12 = 7 + 5$), 13 $(11 + 2)$. Компьютер позволит вам составить оставшийся список.

Господин P не может найти решение, так как его произведение может быть многими разными способами разложено в произведение двух чисел. Учитывая, что именно знает S, он исключает все пары, сумма которых вычеркнута. У него остается в точности одна пара. Каковы произведения, обладающие этим свойством?

Наконец, господин S получает решение. Следовательно, среди всех пар с суммой s есть только одна пара, дающая произведение с упомянутым выше свойством.

Компьютер нужен, чтобы порождать списки и вычеркивать в них. В конце должна оставаться одна и только одна пара.

Головоломка 16.

Я предлагаю вам решить эту задачу в два приема.

1. Составьте сначала программу по методу Полларда — Брента с «маленькими» числами, иначе говоря, такими, что машина представляет их без округления или усечения. Это зависит от машины. Я на своей машине могу получить около $8 \cdot 10^6$, что немного. Возникают еще некоторые сомнения, как только принимаются во внимание деления...

Чтобы узнать, становится ли последовательность периодической, вы можете ограничиться рассмотрением разностей $a_i - a_j$, где i и j меняются в соответствии с вполне определенными законами. Вам следует рассматривать

н.о.д. этих разностей и n. Это невыполнимо для каждой разности и потребует много времени.

Идея в том, чтобы образовать произведение на некоторое число этих разностей по модулю n, а затем брать н. о. д. этих разностей и n. Если одна из этих разностей имеет с n н. о. д., отличный от 1, то для произведения будет выполняться то же самое. Выбор числа членов для участия в произведении предоставляется вашему усмотрению. Если членов слишком мало, то вы вычисляете много н. о. д. и замедляете метод. Если членов много, то вы делаете ненужные операции: вы долго ждете перед тем, как обнаружить делитель...

2. Если эта первая программа уже готова, переходите к гораздо большим числам. Нужно выполнить следующие операции:

произведение двух чисел по модулю n,

н. о. д. двух чисел, числа n и числа, меньшего n.

Настоящая трудность — это произведение по модулю n. Так как к ней часто обращаются, то она должна быть оптимальной...

Может оказаться опасным пускаться в этот метод Полларда, не зная, является ли исследуемое число составным. Используйте для этого тест Ферма.

В этом единственную трудность представляет возведение x в степень $n - 1$ по модулю n.

Следовательно, пусть нужно вычислить $y = x^p$.

Примем следующую индуктивную гипотезу: искомый результат имеет вид $y = u^k w$.

Если k есть нуль, то $u^k = 1$ и потому $y = w$, и все закончено.

Если k не нуль и если k четно, то $u^k = u^{2(k/2)} = (u^2)^{k/2}$. Заменяя u на $u * u$ и k на $k/2$, возвращаемся к общей ситуации.

Если k нечетно, то $u^k = u * u^{2((k-1)/2)}$,

$w * u^k = (w * u) * (u^2)^{(k-1)/2}$.

Заменим w на $w * u$, u на $u * u$ и k на целую часть от $k/2$.

Все это должно проделываться по модулю n. Операции над k не содержат трудностей. Если числа достаточно малы, то вы действуете обычными умножениями или делениями.

Если же числа не являются достаточно малыми, то все сводится к предыдущему случаю. Но у вас здесь есть элемент ответа. Я уже говорил вам, как можно вычислить

$y = x^p$ с помощью бинарного разложения p, выполняя умножения только по модулю n. Переделайте то же рассуждение для $y = p * x$, заменяя возведение в степень умножением, а умножение — сложением. Предположите, что результат имеет вид

$$y = k * u + w.$$

Если k четно, то $k * u$ $(k/2) * (u + u)$, и т. д.

Сложения нужно делать по модулю n, что не требует, впрочем, операции деления...

Я на своем компьютере получил отличные результаты для теста Ферма. А метод Полларда—Брента еще остается очень медленным. Работайте надежно. Можно ли пользоваться программой, в правильности которой вы не уверены?

Головоломка 17.

Подсказка: эта программа сообщает, делится ли n на b.

Головоломка 18.

Снова подсказка: эта программа выводит НЕТ, если n не является точным квадратом; в противном случае она выводит квадратный корень из n. Но это из области бесполезных подсказок. Как вы сможете показать, что эта программа действительно делает то, что я анонсировал? Испытав ее? Вы можете испытать все целые?

По индукции? Почему бы и нет? Напишите мне, если получится...

Головоломка 19.

Не пренебрегайте крохами информации, которые можно извлечь из текста программы. Вполне правдоподобна гипотеза, что eps — параметр, характеризующий точность, маленький и потому вещественный. Следовательно, p и q, и — вследствие этого — a и b имеют хорошие шансы оказаться вещественными. Примите это как гипотезу, касающуюся типа данных и результата.

Вы не можете исследовать плоскость a, b, чтобы увидеть, что же именно вычисляет эта программа. Но можно сделать несколько простых замечаний. Пусть $f(a, b)$ — значение, вычисляемое программой.

Вы без особых усилий сумеете показать, что

$$f(a, b) = f(b, a),$$
$$f(ac, bc) = cf(a, b)$$

и вследствие этого

$$f(a, b) = bf(a/b, 1).$$

Но $g(x) = f(x, 1)$ — функция только одного аргумента. Можно ограничиться областью $x \geqslant 1$. Я написал программу, вычисляющую g (простой и очевидный вариант предыдущей программы), а затем вычислил g для

$x = 1, 2, 3, \ldots, 10,$
$x = 1.1, 1.2, 1.3, \ldots, 1.9.$

Природа функции g становится очевидной, если исходить из этой таблицы. Уразумев, что именно нужно доказать, мы справимся с этим без труда.

3. ИГРЫ БЕЗ СТРАТЕГИИ

Игра 6.

Единственная задача: считать белые шашки. На самом деле, черные можно получить, сравнивая шашку за шашкой в тайной комбинации и в комбинации, предложенной игроком.

Для подсчета белых шашек у вас есть много возможностей.

1. Во время подсчета черных шашек удалите из тайной комбинации и из комбинации, предложенной игроком, находящиеся в соответствии элементы (имеющие одинаковые значения и одинаковые места). Затем для каждого из элементов, оставшихся в предложенной комбинации, посмотрите, участвует ли он в тайной комбинации, и если да, то учтите его белой шашкой и удалите его из тайной комбинации.

Этот метод требует, чтобы вы создали копию тайной комбинации. Это стоит не слишком дорого...

2. Для каждого из возможных значений шашек (6, если есть 6 цветов) подсчитайте число шашек этого цвета в тайной комбинации и в предложенной комбинации. Меньшее из этих двух чисел равно сумме белых и черных шашек, отвечающих этому цвету (почему?).

Так как вы нуждаетесь в подсчете по цветам шашек тайной комбинации и так как эти величины не меняются в течение партии, то вам может оказаться полезным сделать этот подсчет до запрашивания первой комбинации игрока, а затем сохранять его в виде таблицы...

Игра 7.

Для программирования нет совершенно никаких трудностей. Действительно, от вас требуется принять только одно решение: как вы представите игру в вашей программе?

У вас много возможностей.

1. Никакого внутреннего представления игры. Скорость движения машин и интервалы между машинами суть постоянные величины, которые вы можете поместить в таблицу или каждый раз пересчитывать.

Вы можете задать себе с помощью маленькой таблицы положение крайней левой машины на каждой строке. Все остальное можно отсюда вывести. Но вам придется проделать немало вычислений к моменту вывода на экран.

2. Можно сохранять каждую строку в памяти в виде цепочки символов (за исключением, быть может, первых символов, крайних слева). Тогда вывод на экран очень прост. Передвижение машин влево можно получить, удаляя заданное число символов в начале цепочки и добавляя столько же символов справа, соблюдая условия правильного расположения стрелок на правильных местах.

Вам не нужно экономить ни время вычисления, ни объем памяти. Выберите решение, которое, как вам кажется, проще всего реализовать...

Игра 8.

Перемещение шадока почти не составляет проблемы: вы читаете данную в ответ цепочку символов. Если она содержит П, то абсцисса шадока увеличивается на 1, и т. д.

Перемещение многочисленных гиби немного более сложно. Для каждого из гиби нужно найти ближайший к нему цветок. Если у вас есть таблица, задающая положения цветов, то вы ее полностью перебираете и отыскиваете ближайший. Если у вас такой таблицы нет, то нужно вертеться кругами вокруг каждого из гиби, чтобы найти ближайший цветок. Первое решение кажется лучшим.

Но если у вас есть таблица цветов, то нужно, по ходу движения гиби, исключать один за другим уже сорванные цветки (как вы собираетесь это сделать?). Здесь есть и некоторое преимущество: цветок может быть сорван только одним из гиби. Как только это произошло, цветок больше не привлекает других гиби. Когда время сбора плодов истекает, вы знаете, сколько цветков исчезло, и вы случайным образом добавляете столько же цветков в игру.

Вам нужно еще решить, сохраняете ли вы в памяти образ игры (например, в форме цепочки символов или таблицы цепочек, по цепочке на каждую строчку, или двумерной таблицы, дающей содержание каждого поля) или же вы сохраняете только координаты активных элементов

(шадока, гиби и цветов). Не принимайте решение наугад. Тщательно обдумайте преимущества и неудобства каждого решения как с точки зрения результатов, так и с точки зрения простоты программирования (что должно быть определяющим критерием).

Игра 9.

Мало что можно добавить. Единственная трудность — движение убийц. Они должны стремиться приблизиться к игроку, но это не всегда возможно. Вы легко можете определить направление наилучшего перемещения: это направление, которое уменьшает скачок координат убийцы и игрока. Если такое перемещение возможно, то убийца перемещается. Если оно невозможно, то следует испытать близкие перемещения.

Вот способ действовать: задайте таблицу, определяющую возможные перемещения, помеченные индексами от 1 до 8. Желательное перемещение соответствует некоторому элементу таблицы, скажем, элементу с номером k. Испытаем тогда перемещение k, затем, если оно невозможно, перемещения $k-1$ и $k+1$, затем $k-2$ и $k+2$. . . Когда индекс становится нулем, мы его заменяем на 8. Когда он становится больше восьми, мы заменяем его на 1, что сводится к организации таблицы по круговому списку. Внимание, это допустимо только в случае, если вы правильно упорядочили перемещения в таблице...

Как и в предыдущих играх, тщательно продумайте способ представления игры, задайте всю необходимую информацию для упрощения программирования. Не позволяйте себе увлечься оптимизацией времени вычислений или объема требуемой памяти. Эта игра должна легко поместиться на любом микрокомпьютере. Работайте аккуратно: если то, что вы запрограммировали, не удовлетворяет правилам игры, то ваша игра нечестная.

Игра 10.

Добавить нечего. Это проще, чем в случае убийц. Вы определяете направление перемещения, которое наилучшим образом приближает робота к игроку (перемещение, которое уменьшает скачки обеих координат). Вы перемещаете робота в этом направлении. Если он попадает в расщелину, то он исключается из игры. Если он попадает на поле другого робота, то он также исключается. Используйте генератор случайных чисел, чтобы решить, помещаете ли вы в углы новых роботов. Например, если случайное число меньше 0,7, то вы ставите нового робота; в про-

тивном случае вы не делаете ничего. Это означает, что есть 7 шансов из 10 увидеть появление нового робота...

Игра 11.

Никаких особенных трудностей, если не считать тех, которые связаны с рисунком дороги и положением препятствий. У вас много способов представить игру. Если вы используете таблицу, то факт перемещения фигуры очевидно обязывает вас производить сдвиги. Если вы используете цепочки символов, то дело упрощается. Вы можете, например, состыковывать (конкатенировать) различные строчки (вначале — пробелы, 4 знака (точка, звездочка, 0), код окончания строки) в единую строку, которую вы выводите на экран кусок за куском. Чтобы сдвинуть фигуру, вы убираете некоторое количество кусков в начале (определяемое скоростью) и добавляете столько же в конце. Но это неприемлемо, если ваш компьютер не допускает длинных цепочек (счастливы обладатели LSE!).

Чтобы заставить дорогу повернуть, вы изменяете на 1 число пробелов в начале. Но не выбирайте случайным образом одно из трех чисел: —1, 0, 1. (Технически это легко. Вы выбираете случайным образом число в интервале (0, 1), скажем x, а затем берете целую часть от $(3 * * x)$, уменьшенную на 1.) Если вы сделаете так, то дорога останется приблизительно прямой с маленькими колебаниями влево или вправо. Задайте фактор поворота t, принимающий значения —1, 0 или 1. На каждой новой строке вы увеличиваете на t число пробелов в начале. Чтобы изменить t, вы выбираете случайное число. Вы задаете постоянную величину a. Если случайное число меньше a, то вы уменьшаете t на 1, и если это действие дает вам —2, то вы полагаете t равным 1. Если, напротив, случайное число больше, чем $1 - a$, то вы увеличиваете t на 1 и если получаете 2, то заменяете его на —1. Параметр a вы подберете экспериментально.

Для размещения тяжелых грузовиков вы можете случайным образом выбирать целое число в интервале длины, большей 4. Если оно примет значения 1, 2, 3 или 4, то вы помещаете грузовик в соответствующий ряд, а если оно примет большее значение, то препятствия нет. Чем больше выбранный исходный интервал, тем меньше шансов для появления грузовика. Подберите этот параметр экспериментально.

Игра 12.

Итак, мы покончили с «маленькими играми». Все предыдущие требовали лишь немного умения программиро-

вать и немного ловкости. С другой стороны, они требовали большой тщательности, и только хорошие программисты могли сделать из них что-нибудь красивое и приятное. Ну, а в этой игре, по моему мнению, нужно действовать более мощными методами.

Тщательно проанализируйте способ создания комбинации, исходя из 6 шашек. Начало всегда одно и то же.

Вы выбираете две шашки, скажем a и b, и соединяете их одной из операций:

$$a + b, \quad a - b, \quad a * b, \quad a : b.$$

Сложение возможно всегда. Что касается вычитания, то с ним дела обстоят так же, если договориться, что мы всегда вычитаем меньшее из большего (это относится к правильному наименованию чисел, или — что то же — к взятию той из двух операций $a - b$ или $b - a$, которая дает положительный результат). Заметим, однако, что если $a = b$, то знак «—» выбирать нельзя.

$a * b$ можно вычислять только тогда, когда ни один из двух сомножителей не равен 1.

$a : b$ ориентировано (как и вычитание). Число b не должно быть равно 1. Остаток при делении должен быть нулевым.

Все это не очень трудно запрограммировать. Вы случайным образом выбираете две шашки. Затем вы случайным образом выбираете знак операции, а если его нельзя использовать — вы повторяете розыгрыш знака. В конце концов вы всегда получите хороший знак...

Теперь вы получили промежуточный результат. Вы можете решить остановиться, а затем выбрать случайным образом недостающие шашки, которые не участвовали в счете:

7 * 75 = 525 8 3 1 10;

вы выводите на экран

1 3 7 8 10 75 найдено: 525.

Вы можете выбрать новую шашку и скомбинировать ее с предыдущим результатом

525 — 8 = 517.

Вы снова получите промежуточный результат.

Вы можете выбрать две шашки и скомбинировать их:

3 * 7 = 21.

Тогда вы получите два промежуточных результата:

$7 * 75 = 525$; $3 * 7 = 21$.

Если у вас два промежуточных результата, то появляется много возможностей:

— все 6 шашек уже выбраны. Вы комбинируете между собой два промежуточных результата и получаете вашу окончательную комбинацию;

— даже если не все 6 шашек использованы, вы можете скомбинировать между собой два промежуточных результата и снова получить один-единственный результат.

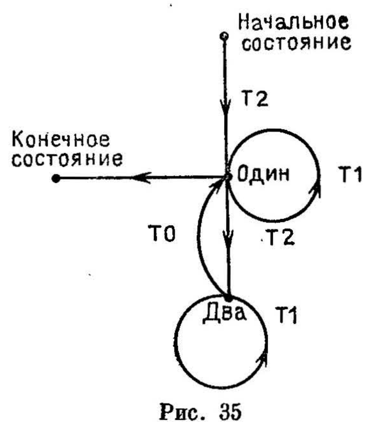


Рис. 35

Но вы можете также выбрать новую шашку и скомбинировать ее с одним из двух промежуточных результатов. Вы снова получите два промежуточных результата.

Таким образом, вы получаете то, что называется конечным автоматом. Есть четыре возможных состояния:

начальное состояние,

состояние ОДИН, в котором у вас есть один (и притом единственный) промежуточный результат.

состояние ДВА, в котором у вас есть два промежуточных результата,

конечное состояние, в котором у вас есть результат, который вы рассматриваете как достигнутую цель.

В вычислениях участвуют три операции.

T2: выбрать случайным образом две шашки и соединить их случайным образом выбранным знаком, чтобы получить промежуточный результат;

T1: случайным образом выбрать шашку и соединить ее случайным знаком с промежуточным результатом;

T0: соединить два промежуточных результата между собой случайным образом выбранным знаком.

Рисунок 35 дает граф этого автомата, где стрелки показывают операции, которые нужно выполнить, чтобы перейти от одного состояния к другому. Ваша программа должна реализовать этот автомат, причем переходы должны выбираться случайным образом, если это возможно,

Вы теперь знаете все. Конечные автоматы часто встречаются в программировании. Запомните этот пример, он имеет очень широкую область применения...

Игра 13.

Проблема наиболее длинного пути взятия является типичной возвратной задачей. Когда лиса находится в некотором положении, нужно испытать 4 возможных направления и для каждого из них увидеть, есть ли курица и свободно ли следующее за ней поле. Это легко!

Если вы не обнаружили никакого возможного взятия, то все закончено.

Если вы обнаружили возможное взятие, то результат есть наиболее длинное взятие, возможное при этом новом исходном положении, увеличенное на 1.

Но вы можете также действовать итеративным способом. Вы делаете первое взятие и продолжаете дальнейшие исследования, исходя из этого поля прибытия. Нужно испытать все возможности. Вы снова получаете, таким образом, тип задач, известный по головоломке 8. Упорядочьте четыре направления перемещения. Вы исходите из некоторого положения с направлением перемещения $i = 1$.

Если все четыре направления испытаны, то все закончено.

В противном случае вы смотрите, возможно ли взятие в направлении i:

— если невозможно, то вы увеличиваете i на 1 и возвращаетесь для нового цикла;

— если возможно, то вы выполняете это взятие, оказываетесь в новом положении и начинаете заново, исходя из него.

Внимание: нужно иметь возможность отменять сделанные вами взятия, потому что они происходят в рамках исследования... Это требует некоторой ловкости. По этой причине рассматриваемая игра — не из самых легких.

Остальное вы исследуете совершенно самостоятельно.

Игра 14.

Ничего нового с точки зрения программирования, за исключением того, что нужно исследовать восемь направлений перемещения вместо четырех.

4. ИГРЫ СО СТРАТЕГИЕЙ

Игра 16. Числа Спрага — Грюнди

В большинстве нижеследующих игр два игрока делают ходы по очереди, и выигрывает тот, кто достигает н-

которой намеченной в начале игры позиции. В той игре, которую мы обсуждаем сейчас, позиция может быть полностью охарактеризована числом оставшихся спичек, и выигрывающая позиция соответствует числу спичек, равному нулю. Спраг и Грюнди предложили (соответственно в 1936 и 1939 годах) связывать с каждой игровой позицией неотрицательное целое число следующим образом:

— выигрывающей позиции вы сопоставляете 0;

— данной игровой позиции вы сопоставляете наименьшее неотрицательное целое, отличающееся от чисел, связанных с позициями, которые могут быть достигнуты, исходя из данной.

Образуем числа Спрага — Грюнди для этой игры.

Позиции 0 сопоставляется число 0, SG (0) = 0.

Исходя из 1, можно получить 0 (поскольку мы имеем право удалить одну спичку *). Следовательно, SG (1) — наименьшее неотрицательное целое, отличное от 0, или SG(1) = 1. Исходя из 2, можно получить 1 и 0. Следовательно, SG(2) — наименьшее неотрицательное целое, отличное от 0 и 1, поэтому SG(2) = 2.

Так как можно удалять спички вплоть до 6, то точно так же имеем

SG(3) = 3, SG(4) = 4, SG(5) = 5, SG(6) = 6.

Предположим теперь, что имеется 7 спичек. Можно удалить от 1 до 6. Поэтому в результате можно получить от 6 до 1 спичек, но не 0. Число SG(7) — наименьшее неотрицательное целое, отличное от 1, 2, 3, 4, 5, 6. Следовательно, это 0.

SG(7) = 0.

А теперь из 8 можно получить от 2 до 7, поэтому SG(8) — это не 2, не 3, . . ., не 6 и не 0, поэтому оно равно 1.

SG(8) = 1.

Теперь вы можете установить общий закон:

SG (p) = остаток от деления p на 7.

Как же выигрывать?

Если вы после своего хода можете оставить кучу, для которой число Спрага—Грюнди равно 0, то ваш против-

---

*) Важно и то, что никаких других позиций, кроме 0, из 1 получить нельзя.— *Примеч. ред.*

ник не сможет достичь ситуации с числом нуль, поскольку по определению число, которое он оставит, отлично от исходного числа. Поскольку он не сможет достичь ситуации p с SG $(p) = 0$, то он и не может выиграть. Ему придется оставить вам ситуацию с SG $(p) \neq 0$, исходя из которой, вы всегда сможете получить ситуацию с числом Спрага—Грюнди, равным нулю. Следовательно, вам нужно оставлять вашему противнику число спичек с числом SG, равным нулю, иначе говоря, число спичек, кратное 7.

Одно из двух: либо ваш противник не знает этого правила и играет «по нюху»; при первой возможности вы оставляете ему кратное 7 и из ежовых рукавиц не выпускаете; либо он знает правило и ходит первым: он достигает кратного 7. Вы не сможете выиграть, если он не рассеян или не сделает ошибки в счете. Но компьютер не рассеян и не делает ошибок в счете (если ваша программа верна)...

Игра 17.

Выигрывающее положение — 31 декабря. Возьмите листок бумаги в клетку. Расположите по абсциссе месяцы, а по ординате дни. Так как 31 декабря выигрывает, то вы обозначаете эту точку числом Спрага—Грюнди 0. Из каждого дня декабря можно получить 31, но также и любой другой последующий день. Поэтому вы приписываете значение 1 дате 30 декабря, значение 2 дате 29 декабря, и т. д. То же для любого 31 числа; из него можно получить 31 число всех последующих месяцев. Поэтому 31 октября получает 1, 31 августа 2 и т. д.

После этого вы можете закончить значение таблицы и приписать число Спрага — Грюнди всем дням года. Вы увидите также появление дней со значением 0, которые являются выигрывающими днями. Напоминаю вам правило: каждому игровому положению приписывается наименьшее неотрицательное целое значение, отличное от значений тех положений, которые можно получить, исходя из данного, т. е. в настоящем случае — от значений тех положений, которые расположены правее, и тех, которые расположены ниже.

Закон заполнения таблицы достаточно сложен; и я не пытаюсь вам его сформулировать. Как только октябрь заполнен, появляется простая закономерность, которая дает соотношение между номером дня и номером месяца для выигрывающих положений.

Даже если вы мало знаете современную математику, вы слышали разговоры об отношении эквивалентности.

Все выигрывающие положения эквивалентны. Игровое положение задается парой $\partial, м$, где ∂ — номер дня, а $м$ — номер месяца. Следовательно, вы должны найти такое отношение эквивалентности для пар натуральных чисел, чтобы

$\partial, м'$ было не эквивалентно $\partial, м$ при $м \neq м'$, и

$\partial', м$ было не эквивалентно $\partial, м$ при $\partial \neq \partial'$.

Наконец, для выигрывающей позиции $\partial, м$ должно быть эквивалентно 31, 12. Что-то похожее на это можно видеть в программах лицеев...

Я прекрасно понимаю, что календарь осложняет все, поскольку длина месяца не постоянна и зависит от $м$, причем к тому же с непростым законом изменения. Но, к счастью, оказывается, что это никак не сказывается на этом замечательном отношении эквивалентности.

После всего сказанного вы должны выпутаться из этой задачи...

Игра 18.

Эта игра — производная от средневековой игры. Сначала попытайтесь достичь 50 с точностью до кратного 7. Но как только все четыре карты, имеющие одинаковое значение, оказываются использованными, так ситуация сразу меняется. Вот пример начала партии.

Я беру туза, компьютер тоже. Сумма 2.

Чтобы получить 8, я беру 6. Компьютер берет туза. Сумма 9.

Чтобы получить 15, я снова беру 6.

Компьютер берет последнего туза. Сумма 16.

Теперь остаются следующие карты:

2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6

Так как тузов больше нет, то числа Спрага — Грюнди изменились *). Теперь из 49 больше нельзя получить 50.

SG (50) = 0, SG (49) = 0.

---

*) Читатель может вернуться к определению чисел Спрага—Грюнди и убедиться, что эти числа определяются на множестве игровых позиций раз и навсегда, исходя из правил игры, и, разумеется, не могут меняться в процессе разыгрывания конкретной партии. Что же является позицией в этой средневековой игре? — Позицией является состав выложенных на стол карт, а также их значения: сколько карт на столе имеет значение 1, сколько карт имеет значение 2, и т. д. Сумма, набранная игроками в данный момент, равна 84 минус сумма значений карт на столе. Что же имеет в виду автор книги, когда он пишет SG (50)? Почему он приписывает число Спрага—Грюнди не позиции, а сумме карт этой

Из (48) можно получить 50. Поэтому SG (48) = 1.

Из 47 можно получить 49 и 50, но не 48. Поэтому SG (47) = 1.

Теперь положения, имеющие нулевое SG,— это

42 41 34 33 26 25 18 17

Поэтому я могу взять 2, чтобы достичь 18.

Стратегия усложняется, поскольку числа Спрага — Грюнди полностью меняются при удалении каждой карты. Но это как раз и благоприятствует компьютеру. Если он не может достичь выигрывающего положения, он берет карту, оставшуюся в наименьшем количестве экземпляров. Каждый раз, когда тот или иной тип карт исчерпывается, компьютер пересчитывает заново числа Спрага — Грюнди.

Мне придется переписать мою программу в соответствии с этой стратегией.

Игра 19. Ним-сумма.

Для меня эта игра — своего рода педагогический вызов. Я чрезвычайно раздражен тем, что все, кто излагает эту игру, ведут себя одинаково: известно, что выигрывающей стратегией является следующая... Почему она выигрывает? Откуда она вообще взялась эта стратегия?

Выписать числа Спрага — Грюнди очень трудно.

Попытаемся найти несколько выигрывающих положений:

Если к моменту своего хода я обнаруживаю только одну спичку, то я выигрываю.

Если я обнаруживаю единственную кучку, то я тоже выигрываю.

Если, кроме одной кучки, ничего больше нет, то можно положить SG (0) = 0 (я выигрываю, я взял последнюю спичку), вследствие чего SG $(n) = n$.

Предположим теперь, что у нас две кучки. Если я оставляю две кучки, в каждой из которых по одной спичке, то я обязательно выигрываю: мой противник должен взять столько спичек, сколько он хочет, но — только из одной

позиции? Дело в том, что для всех позиций с набранной суммой 50 число Спрага—Грюнди одинаково и равно 0. Это и позволяет написать равенство SG (50) = 0. А что могло бы значить SG (49)? Если бы все позиции с суммой 49 имели одинаковое число SG, мы бы обозначили его SG (49). Но, увы! Разные позиции с суммой 49 имеют разные числа Спрага—Грюнди. Так что автор книги дальше рассуждает о несуществующих вещах. Я из этих рассуждений ничего полезного извлечь не смог (кроме подозрения, что у автора нет работающей программы, играющей в 24 карты).— *Примеч. ред.*

кучки. У него нет выбора, он может только взять одну из спичек, после чего я возьму последнюю спичку и выиграю.

Если я оставляю две одинаковые кучки по n спичек в каждой, то у моего противника две возможности:

— взять целиком одну из кучек, я возьму другую и выиграю;

— взять часть одной из кучек и оставить в ней n' спичек. Я возьму столько же из другой, оставляя ситуацию n', n'. По индукции — я на пути к победе.

В наиболее общем случае ситуация характеризуется p целыми числами (p — число кучек). При каждом ходе изменяется одно и только одно из этих неотрицательных целых чисел и оно заменяется меньшим неотрицательным целым числом, которое может быть и нулем. Если мы исходили из выигрывающей ситуации, то новая ситуация не является выигрывающей. Если ситуация не является выигрывающей, то всегда можно, уменьшая одно из чисел, получить выигрывающую ситуацию (по крайней мере, если выигрывающая стратегия существует *)...).

Попытаемся охарактеризовать числа с помощью их цифрового представления. Изменить число — значит, изменить представляющие его цифры. Если использовать десятичное представление, то у нас в наличии 10 возможных цифр и их изучение затруднительно. Возьмем двоичное представление, для которого есть только две возможные цифры: 0 и 1. Уменьшение числа изменяет по крайней мере одну цифру этого числа, так что есть по крайней мере одна цифра 1, замененная на 0, или 0, замененный на 1. Этого должно хватить для того, чтобы заставить перейти от выигрывающего положения к проигрывающему положению. Число 0 встречаться не должно, поскольку пустые кучки, характеризующиеся нулевыми значениями, просто не считаются кучками. Характеризация выигрывающего положения должна быть поэтому связана с единицами различных чисел, записанных в двоичной системе.

Если есть две кучки с одинаковым числом спичек, то ситуация является выигрывающей. Следовательно, каково бы ни было число единиц в двоичном представлении

---

*) Можно доказать, что в играх, подобных игре Нима и обычным шахматам,— в играх с полной информацией — выигрывающая стратегия всегда существует. Это — относительно простая теорема. Другое дело, что эта выигрывающая стратегия может быть не очень простой (Ним) или вообще еще не открытой (шахматы).— *Примеч. ред.*

каждого числа, положение является выигрывающим, если в каждом разряде наши два числа имеют либо 0, либо две цифры 1.

Первые выигрывающие комбинации с тремя кучками имеют вид

1, 2, 3, или в двоичной записи 01 10 11,

1, 4, 5, или в двоичной записи 001 100 101

Опять в каждом разряде наши три числа имеют либо 0, либо две цифры 1. Я разобрал достаточно случаев, чтобы подвести вас к результату К. Бутона (1902): положение является выигрывающим, если в каждом двоичном разряде суммарное число 1 двоичных представлений числа спичек в каждой кучке четно.

Совершенно очевидно, что нужно совершить прыжок для перехода от случая двух куч или первых примеров в случае трех куч к наиболее общему случаю. Тут требуется выдумка или изобретение. Следует иметь мужество признать, что некоторые люди имеют настоящий талант изобретать или открывать то, что совершенно не очевидно, и не всегда можно потом сказать: да никакой заслуги в этом нет, это было очевидно. Нет, это остается тайной, и преподавателя раздражает, если он оказывается вынужден давать результат, который нельзя легко «переоткрыть».

Назовем Ним-суммой двух целых чисел p и q число, которое вычисляется следующим образом:

p и q записываются в двоичной системе;

сложение выполняется поразрядно, по следующему правилу:

$$0+0=0, \quad 0+1=1+0=1, \quad 1+1=0$$

(сложение без переноса в следующий разряд).

Рассмотрим игру, образованную объединением n независимых игр, каждая со своими собственными правилами. Игра проходит в кучке 1 по правилам $R1$, в кучке 2 — по правилам $R2$, . . . в кучке n — по правилам Rn. В каждой кучке мы располагаем числом Спрага — Грюнди, зависящим от числа спичек в этой кучке. Число Спрага — Грюнди есть Ним-сумма чисел Спрага — Грюнди в каждой кучке... *) Красиво, не правда ли?

---

*) В частном случае, когда в каждой кучке игра идет по правилам игры Нима, число Спрага—Грюнди каждой кучки равно просто числу спичек.— *Примеч. ред.*

Обратимся к программированию обычной игры города Нима (одно и то же правило для всех кучек: можно брать столько спичек, сколько пожелаешь, но не меньше одной). Вам нужно вычислить Ним-сумму данной ситуации. Если она равна нулю, то у вас нет шансов: ситуацию придется изменить и она перестанет быть выигрывающей. Вы можете, например, взять одну спичку из самой большой кучи: это — способ замедлить конец, и вы всегда можете ожидать, что ваш противник допустит ошибку...

Если же эта сумма не равна нулю, то это в точности означает, что есть разряды, в которых при двоичном представлении единицы встречаются нечетное число раз. Рассмотрим крайний левый из таких разрядов. Нужно уменьшить число единиц в этом разряде. Выберите кучку, содержащую единицу в этом разряде (все равно какую: взять ли самую большую, первую или последнюю...). Нужно уменьшить эту кучку на «эту» единицу. Кроме того, в любом другом (расположенном правее) разряде, где стоит нечетное число единиц, нужно

если в данной кучке в этом разряде стоит 1, удалить ее;

если в данной кучке в этом разряде стоит 0, заменить его на 1.

Это дает вам новое число спичек в этой кучке.

Я видел в некоторых книгах программы для игры Нима, в которых после обнаружения ситуации с ненулевой Ним-суммой испытывались все возможные конфигурации, чтобы найти конфигурацию с нулевой Ним-суммой. Над кем они смеются?

В игре Нима вам нужно для каждого хода, делаемого компьютером, получить двоичное представление числа спичек в каждой кучке. Вам следует решить, будете ли вы пересчитывать его при каждом ходе или вы будете сохранять различные представления, так как имея единственное представление, вы после каждого хода должны изменять его...

Я полагаю, что вы знаете, как получать двоичное представление числа. Пусть

$$n = a_p 2^p + a_{p-1} 2^{p-1} + \dots + a_2 2^2 + a_1 2 + a_0.$$

Если разделить n на 2, вы получаете в остатке a_0, крайнюю справа цифру двоичного представления, и частное

$$a_p 2^{p-1} + \dots + a_2 2 + a_1,$$

которое также является двоичной записью целого числа, получаемой из предыдущей записи вычеркиванием ее крайней правой цифры. По индукции (или, что то же самое, рекурсивно или итеративно) вы получите все двоичные цифры числа n справа налево.

Восстановление значения числа, исходя из двоичных цифр, производится в обратном порядке, слева направо. Сначала вы вычисляете

$$x_0 = a_p,$$
$$x_1 = 2x_0 + a_{p-1} = 2a_p + a_{p-1},$$
$$x_2 = 2x_1 + a_{p-2} = a_p 2^2 + a_{p-1}2 + a_{p-2},$$

и т. д. Последнее x есть искомое значение.

Игра 20.

Об этой игре я вам больше ничего не скажу. Совершенно необходимо, чтобы вы хотя бы время от времени работали. Впрочем, если я вам ничего не говорю, то дело, вероятно, в том, что я вам уже достаточно рассказал. Это — новая головоломка: выясните, почему у меня нет нужды что-либо вам еще говорить...

Игра 21.

Не протестуйте, я вам помогу... Что бы вы без меня делали? Но кстати, нужно быть честным — я был вдохновлен книгой Роуза Болла [BAL].

В начале игры у вас одна-единственная строка: Спраг—Грюнди... По прошествии некоторого времени она разбивается на много строк, и связанное с ними число Спрага — Грюнди есть Ним-сумма чисел Спрага—Грюнди для каждой строки. Следовательно, нужно вычислить числа Спрага—Грюнди для одной строки, и этого будет достаточно. Вот начало:

0 SG (0) = 0

Из 1 вы достигаете 0: SG (1) = 1.

Из 2 вы достигаете либо 1, либо 0. Поэтому SG (2) — наименьшее целое неотрицательное, отличное от 0 и 1; следовательно, SG (2) = 2.

Исходя из 3, вы можете получить либо одну строку с 2 спичками (SG = 2), либо одну строку с одной спичкой (SG = 1), либо две строки по одной спичке в каждой (удалив среднюю спичку). Но число SG (1,1) есть Ним-сумма 1 и 1 и потому равно нулю. Следовательно, SG (3) равно трем. Таким же образом вы находите

0 1 2 3 1 4 3 2 1 4 2 6 4 1

Р. К. Ги доказал, что начиная с 71, эта последовательность становится периодической с периодом 12. Я не представляю себе, для чего это может быть вам нужно — разве что, если это доставит вам удовольствие, чтобы передоказать его.

Задайте компьютеру таблицу первых чисел Спрага—Грюнди, снабдите его Ним-суммой. Остальное просто.

Игра 22.

Каждая вершина может быть связана с 5 другими, что создает $6 \times 5 = 30$ связей. Но каждая из них считается

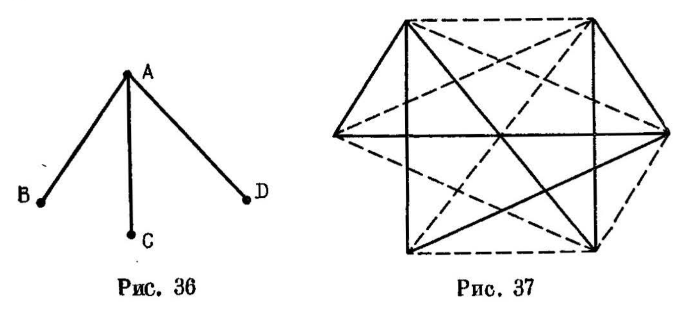


Рис. 36 Рис. 37

дважды (связь между A и B и между B и A). Поэтому есть 15 отрезков, которые нужно провести. Если игра полностью сыграна и все пути пройдены, то у одного из игроков на чертеже должно быть 8 отрезков (у того, который начинает). Они связывают 16 вершин, и поскольку в игре участвует только 6 вершин, то имеется хотя бы одна вершина, из которой выходят три отрезка. Пусть B, C и D — достигаемые этими отрезками вершины (см. рис. 36). Либо этот игрок прошел один из путей связывающих эти вершины, и тогда он проиграл. Либо он ни одного из них не провел, и тогда их провел его противник и противник проиграл...

Может оказаться, что проведено 14 отрезков, не образующих треугольников (как показано на рис. 37).

В этой позиции можно быть уверенным, что кто начинал, тот и проиграет, поскольку нет возможности свести партию вничью. Число возможных комбинаций очень велико, и вы не можете ожидать, что компьютер перепробует все возможные комбинации, прежде чем принять решение. Нужно отказаться от комбинаторных соображений и играть эвристически. Первый ход, если его делает компьютер, не важен: все прямые равноценны. После этого у компьютера остается не более 14 возможных линий, и он

их все исследует. Каждой из них он сопоставляет вес: 0, если эта линия завершает треугольник из его линий, и он тем самым проигрывает; 1, если эта линия завершает треугольник для его противника, так как она оставляет противнику шанс проиграть; максимальный вес, если эта линия связывает еще не использованные вершины. Когда все линии испытаны таким образом, компьютер делает ход с наибольшим весом. Его стратегия оценит шкалу весов, которые вы будете выбирать.

Игра 23.

В этой игре вы не можете охарактеризовать ситуации числом спичек, оставшихся в кучке, потому что этого недостаточно; нужно еще знать, сколько спичек только что было взято, так как именно это определяет максимальное число спичек, которые вы можете взять. Поэтому нужно определить ситуацию парой

p: число спичек, оставшихся в кучке,
q: число спичек, которое только что было взято.

Положение 0 является выигрывающим, каково бы ни было число спичек, только что взятых, чтобы достичь этого состояния:

$SG(0, q) = 0.$

Исходя из 1, мы всегда проигрываем, поскольку обязаны взять единственную оставшуюся спичку:

$SG(1, q) = 1.$

Если у вас осталось две спички, то всегда можно одну взять и одну оставить, следовательно, $SG(2, q) \neq 1$, или можно взять две и закончить игру:

$SG(2, q) = 2.$

Начиная с трех, выбор меняется.

Для 3, 1 ваш противник может взять 1 и оставить пару 2, 1, следовательно, $SG(3, 1) \neq 2$, либо взять 2 и оставить пару 1, 2, так что $SG(3, 1) \neq 1$. Но большее количество изымать нельзя. Наименьшее неотрицательное целое, отличное от 1 и 2, есть 0:

$SG(3, 1) = 0.$

Если вы оставляете 3, взяв больше, чем одну спичку, то противник может взять и 3, достигая 0 с $SG(0, 3) = 0$, и, следовательно,

$SG(3, q > 1) = 3.$

Все, что от вас здесь требуется — продолжить это изучение достаточно далеко, чтобы дать компьютеру список выигрывающих положений,— и тогда ваша программа будет непобедимой.

Игра 24.

Я много раз излагал нижеследующее различным программистам и каждый раз оставался в недоумении, видя, что они не считают это «очевидным».

Вы играете в «Гениального отгадчика», вы ищете неизвестную комбинацию; чтобы сделать это, вы предлагаете комбинации $c_1, c_2, \ldots, c_k$. Для каждой из них вы получаете ответ о числе белых и черных шашек:

$б_1$ $ч_1$; $б_2$, $ч_2$; ...; $б_k$, $ч_k$.

Следующая предлагаемая комбинация должна быть такой, которая при сравнении с c_1 дает $ч_1$ черных и $б_1$ белых шашек; ...; при сравнении с c_k она должна давать $ч_k$ черных и $б_k$ белых шашек. Почему? Вы ищете неизвестную комбинацию. Но эта комбинация дает при сравнении с комбинацией c_i именно $ч_i$ черных и $б_i$ белых шашек. Бесполезно искать решение вне множества комбинаций, обладающих этим свойством: там его не может быть *).

Следовательно, у вас есть простая стратегия. Допустите, что вы уже каким-то образом выбрали x первых комбинаций, где x фиксировано. Компьютер располагает значениями $ч_i$, $б_i$ для i от 1 до x. Вы предоставляете ему возможность перепробовать все комбинации и запоминать только те, которые дают при сравнении с уже испытанными комбинациями правильные значения черных и белых шашек.

Так как возможных комбинаций много, то нужно попытаться не перебирать их все заново при каждой следующей попытке. Вы можете, например, начать с первой позиции новой комбинации. Вы присваиваете ей первый цвет, а затем смотрите, сколько черных шашек он образует с уже испытанными комбинациями. Если он дает черную шашку с комбинацией, с которой ее не следует давать, то этот цвет нужно отбросить. Когда вы уже нашли под-

---

*) Эти рассуждения безусловно справедливы, если в моем распоряжении остался один-единственный ход — тогда этим ходом я хочу «попасть в десятку», т. е. угадать искомую комбинацию. Если же ход не последний, то моя цель — получить как можно больше информации об искомой комбинации. Может случиться, что для этого выгоднее взять комбинацию вне множества, описанного автором.— *Примеч. ред.*

ходящий цвет для этой позиции, переходите к следующей. Она может дать вам слишком много черных шашек, и это событие очень даже вероятно. Мало таких комбинаций, которые черных шашек не дают совсем, и больше таких, которые дают не более одной. То, что было зафиксировано для первого цвета, не может быть использовано для второго. Но заметьте, что у вас есть и другой случай для отбрасывания: если нужно получить три черных шашки при сравнении с некоторой комбинацией и если первая позиция никакого вклада не вносит, то необходимо, чтобы вторая позиция вносила свой вклад (предполагая, что есть 4 позиции). Действуя таким образом, вы достигаете в конце концов комбинации с правильным числом черных шашек. Тогда нужно проверить белые. Если они принимают нужные значения для всех предложенных комбинаций, то у вас готово новое предложение, и вы получите либо успех, либо новые элементы для сравнения.

Если испытание комбинации потерпело неудачу, то вы переходите к следующей, начиная, если это возможно, с последнего, отступая на одну позицию и испытывая следующий цвет на этой самой позиции.

Для экономии вычислений вы можете быть заинтересованы в том, чтобы сохранить некоторые результаты, полученные во время исследования одной позиции за другой. Но внимание! Когда вы возвращаетесь назад, нужно знать, как определить, что нужно сохранить, а что исключить. Используйте при необходимости изучение «гениального ответчика» (игра 6), чтобы выбрать наилучший способ определения белых и черных шашек.

Игра 25.

Как и при игре Сима, невозможно действовать из комбинаторных соображений, т. е. изучать при каждом ходе компьютера все окончания всех возможных партий. Поэтому нужно взвешивать различные ситуации и делать наилучший ход.

Я предоставляю вам выбор весов...

Игра 26.

Все то же самое. У вас 7 игровых положений. Поэтому ваша программа должна пробежаться по 7 столбцам и для каждого из них вычислить вес игрового положения. Ход определяется положением с наибольшим весом. Если есть два положения с одинаковым весом, предпочтительнее более высокое.

Чтобы определить веса игрового положения, нужно видеть, принадлежит ли оно отрезку из четырех игровых

положении, уже содержащему 3 ваших шашки (тогда именно так и нужно играть: максимальный вес) или 3 шашки противника (максимальный вес минус 1). Если игровое положение находится на пересечении двух отрезков, содержащих по две шашки противника и ничего больше, то оно представляет очень большой интерес для хода. Продолжите этот анализ, и ваша программа будет носить ваш отличительный знак.

Совершенно ясно, что для каждого игрового положения нужно знать состояние всех проходящих через него отрезков. В этой игре есть 50 различных отрезков с 4 положениями. Выбор ясен:

— либо на каждом ходе вы определяете с помощью программы состояние всех отрезков, проходящих через точку;

— либо у вас есть таблица, задающая состояния всех отрезков. В этом случае нужно обновлять данные после каждого хода.

После всего этого вам нужно сделать еще один выбор. Пусть дано игровое положение, нужно узнать список проходящих через него отрезков:

— либо вы определяете его с помощью программы,

— либо вы его задаете с помощью таблицы. Поскольку она не меняется в течение игры, то эта таблица вычисляется раз и навсегда.

Поскольку одно и то же игровое поле может изучаться несколько раз, то, конечно, более выгодно устроить обращение к таблице. Но вам придется преодолеть две трудности: число отрезков, проходящих через данную точку, не постоянно, но меняется от точки к точке, таблицу очень неприятно распечатывать. Я написал верную программу, но я сделал немало ошибок при наборе таблицы и пришлось их исправлять...

Остается способ вычисления состояния отрезка. Я принял следующее соглашение:

— поле, ход на которое невозможен, обозначается 0 (нулем);

— поле, ход на которое возможен, обозначается 1.

Так как нужно изучить отрезки, проходящие через игровое поле, то их наименьшее число 1, но может доходить и до 4. Поэтому нужно быть в состоянии выделять среди них сегмент, содержащий игровое поле и шашку +. Следовательно, придадим такой шашке значение 4. Может появиться отрезок, содержащий . +++ со значением 13, отличающийся от сегмента с игровым полем и шашкой 0.

Поэтому я придаю такой шашке значение 13. В общем, можно взять в качестве значения отрезка сумму значений пометок на этом отрезке. Наконец, нужно задать таблицу, сопоставляющую вес каждому возможному значению отрезка.

В вашу программу входит тогда много данных, но взамен у вас отличное время ответа. Если у вашего компьютера память слишком мала, чтобы иметь возможность сохранить все данные, не храните их и проделывайте вычисления, когда это необходимо. Тогда вы потеряете время на ответ, и выигрыш в пространстве не обязательно будет слишком большим: ведь вы замените данные программой...

5. СТРАТЕГИЯ БЕЗ ИГРЫ (ВЫИГРЫВАЮЩИЕ СТРАТЕГИИ)

Игра 27.

Чтобы найти рекурсивное решение в игре НАДЕВАТЬ, нужно действовать по индукции. Назовем НАДЕВАТЬ (n) решение, которое помещает n шашек на первоначально пустое игровое поле. Предположим, что мы умеем выполнять задание игры НАДЕВАТЬ для p, меньших n.

Как поставить на место последнюю шашку? Мы не можем ее поставить, если это поле не является следующим за первым полем, занятым шашкой. Следовательно, для ее помещения на место нужно, чтобы в игре участвовала одна-единственная шашка, шашка с номером $n-1$. С помощью НАДЕВАТЬ ($n-1$) можно поставить на место все шашки от 1 до $n-1$. Если мы удалим все шашки от 1 до $n-2$, то останется только шашка $n-1$, можно будет поставить шашку n, а затем снова надеть шашки от 1 до $n-2$:

НАДЕВАТЬ(n) = НАДЕВАТЬ($n-1$);
СНИМАТЬ($n-2$); поместить(n); НАДЕВАТЬ($n-2$)

То же самое вы должны проделать и для СНИМАТЬ. Эта запись не учитывает простых частных случаев, позволяющих избежать в этом рекурсивном определении порочного круга: оно должно содержать не рекурсивные случаи. Определение должно включать $n-1$ и $n-2$. Вы можете либо определить игру НАДЕВАТЬ для $n=0$ (ничего не делать) и $n=1$ (поставить первую шашку, что всегда возможно), либо для $n=1$ и $n=2$. Вы сами решите, как лучше сделать.

Но еще более удивительно изучение «итеративной» стратегии для этой игры, т. е. последовательности ходов,

приводящих к выигрышу. Рассмотрим игру НАДЕВАТЬ. Вы увидите, что первый ход предопределен. Используйте тот факт, что ход не должен разрушать то, что было сделано на предыдущем ходе. Вы установите, что

— вы делаете первой шашкой один ход из двух,

— остальные ходы полностью определены,

так что в игре НАДЕВАТЬ нет никакого выбора. Она полностью определена на каждом ходе: делайте единственно возможный не глупый ход...

Для игры СНИМАТЬ есть два способа начать игру:

— удалить сначала шашку 1 (это возможно всегда),

— удалить сначала шашку 2 (это шашка, которая следует за первой шашкой, расположенной на игровом поле).

Никакого другого выбора сделать уже нельзя, все остальное полностью определено. Выясните, как сделать этот первый выбор.

Игра 28.

Есть только одно указание, чтобы помочь вам, если вы не нашли решение: есть промежуточное решение, в котором шашки перемежаются. Вы можете составить сначала рекурсивную процедуру, которая их перемежает, а затем рекурсивную процедуру, которая их заново разделяет. Но вы можете сделать это и итеративным способом...

Игра 29.

Используйте индукцию или ее двоюродную сестру рекурсию. Если у вас на вашем компьютере рекурсивных возможностей нет (бедные владельцы Бейсика...), используйте ее по крайней мере в вашем черновике: хорошая рекурсивная процедура — лучшее из описаний решаемой задачи.

Решите сначала задачу с 8 буквами и 10 полями.

Рассмотрим теперь более общую задачу. Пусть *X* обозначает некоторую последовательность пар *аб* без пустых полей. Используя предыдущий метод (та же последовательность ходов плюс один), перейдите от ситуации

. . абабXабаб

к ситуации

бббб . . Xаааа

затем решите задачу для *X* и отправьте два последних *а* на их место.

Но таким способом вы не охватываете всех возможных случаев. Нужно найти решения в других частных случаях. Вы легко найдете, в каких.

Игра 30.

Это — типичная игра, которая анализируется методом систематического перебора всех возможных решений. Их гораздо меньше, чем может показаться, до такой степени, что в наиболее простых случаях все это выполнимо вручную. Так, для креста на рис. 23 есть (с точностью до симметрий) только три игровых хода.

Если вы поднимете шашку на пересечении двух ветвей креста, то следующие два хода вынуждены и вы проиграли. Если вы спустили шашку до низа креста, то у вас после этого есть выбор между двумя ходами и в любом случае вы проигрываете. Если вы перемещаете шашку на пересечении двух ветвей креста вправо (или влево), то следующий ход вынужден, а затем у вас есть выбор между тремя ходами, два из которых сразу проигрывают, а оставшийся выигрывает.

Тогда без колебаний составляйте:

— либо рекурсивное решение. У меня есть процедура, которая решает задачу с n шашками. Какова бы ни была начальная конфигурация, для любого возможного хода вы этот ход осуществляете и решаете задачу с $n - 1$ шашками;

— либо итеративное решение. Оно отличается от предыдущего только необходимостью восстанавливать игру при возвращении назад. Это приводит вас к вопросу о представлении игры. Возможностей много...

Игра 31.

Поскольку рекурсивное решение тащится по всем книгам, я его вам здесь и предлагаю: это избавит вас от поисков...

Нужно перенести диски со стержня номер $н$ (начального) на конечный стержень номер $к$. Номер запасного стержня x (хранилища) таков, что $н, к, x$ есть перестановка чисел 0, 1, 2, поэтому $н + к + x = 3$. Номер запасного стержня равен $3 - н - к$. Чтобы решить задачу, перенесем $n - 1$ первых дисков со стержня $н$ на стержень x с помощью $H(n - 1, н, 3 - к - н)$.

Затем мы переносим последний диск n с $н$ на $к$, что обозначается

$$P(n, н, к).$$

Эта процедура, которая реализует, например, сооб-

щение

n ИДЕТ С $н$ НА $к$

Наконец, мы переносим $n - 1$ первых дисков с запасного стержня на стержень $к$:

$H(n - 1, 3 - н - к, к)$.

Нужен частный случай, не являющийся рекурсивным. Если диск всего один, то можно сразу перенести его от $н$ к $к$:

$H(p, н, к) =$ ЕСЛИ $p = 1$ ТО $P(1, н, к)$
    ИНАЧЕ $H(p - 1, н, 3 - н - к)$
    $P(p, н, к)$
    $H(p - 1, 3 - н - к, к)$
  КОНЕЦ_ЕСЛИ

Проще некуда. Как же может случиться, что находятся и такие, кому эта процедура внушает опасения? В том ли дело, что они не видят, как на самом деле двигаются шашки? Или дело в том, что они испытывают сомнения в правильности процедуры? Продумайте это решение: если оно составляет для вас задачу, то только потому, что вы не владеете рекурсией, и жаль, что это так...

Число ходов игры легко выводится из этой процедуры. Обозначим через $f(p)$ число ходов, необходимых для игры с p дисками. Из рекурсивной процедуры следует, что

$f(1) = 1$,
$f(p) = 2 * f(p - 1) + 1$.

(Почему?) Исходя из этого, вы можете вычислить $f(p)$ (на самом деле $g(p) = f(p) + 1$ имеет более простой закон построения, чем $f(p)$. Образуйте сначала этот закон, найдите решение, а затем выведите закон для $f(p)$).

Чтобы доказать свойство, касающееся четности дисков, действуйте по индукции по ходу вычислений. Предположите, что это свойство выполняется для $H(p - 1, \ldots)$. Покажите, что от сюда следует его справедливость и для $H(p, \ldots)$.

У вас не получается? Вот дополнительная помощь. Начнем с переноса $p - 1$ дисков на запасной стержень. Пока не передвинут $(p - 1)$-й диск, ни один диск не кладется непосредственно на диск с номером p, и требуемое свойство выполняется. Рассмотрим момент, когда $p - 2$ дисков находятся на одном стержне, диски с номерами $p - 1$ и p — на другом стержне, а третий стержень пуст. Вы пере-

мещаете диск с номером $p - 1$. Тепёрь, поскольку нужно переместить первые $p - 2$ дисков на диск с номером $p - 1$, то диски будут оказываться на диске с номером p. Если мы помещаем диск с номером q на диск с номером p, то для того, чтобы образовать пирамиду дисков с номерами от q до 1 и иметь возможность переместить диск с номером $q + 1$, который отправится на диск с номером $p - 1$. Но требуемое свойство выполняется для $p - 1$ дисков, и поэтому четность диска $q + 1$ не может совпадать с четностью $p - 1$. Следовательно, она совпадает с четностью p. Следовательно, p и q имеют разные четности.

Потренируйтесь в доказательствах такого рода...

Игра 32.

Предыдущее рекурсивное решение имеет ту особенность, что она не включает в ход игры никакого представления этой игры. Если вы хотите представить игру на экране даже символическим образом, вам придется создавать представление игры самому.

Но трудность состоит только в осуществлении видимого представления, потому что нужно учесть все, сказанное выше. Предположим, что нужно выполнить *Р* (*р*, *н*, *к*). Вы знаете, что нужно осуществить движение, которое вводит в игру диск размера *р*, покидающий стержень *н*, с которого он отправляется на стержень *к*. Это означает, что диск *р* находится на вершине стержня *н*, в противном случае его нельзя было бы оттуда взять. Поэтому вы можете не обращать никакого внимания на значение *р*.

Операция *Р* (*р*, *н*, *к*) на самом деле следующая: снять диск с вершины стержня *н* и поместить его на вершину стержня *к*.

Если представить игру в виде 3 строк с помощью последовательностей чисел, то, таким образом, достаточно снять крайнее правое число со строки *н* и присоединить его справа к строке *к*.

Если вы хотите представить стержни вертикально, создайте, кроме того, внутреннее представление с помощью трех цепочек символов и составьте процедуру вывода на экран. Это, как кажется, проще всего. Если вы не любите цепочек символов, используйте три таблицы, но вы не выиграете в легкости.

Игра 33.

Если ваш компьютер допускает рекурсию, заставьте работать рекурсивную процедуру и понаблюдайте за движением дисков. В противном случае выполните вручную рекурсивную процедуру для маленького n (например 4),

что поможет вам наглядно увидеть то, что уже доказано: два диска одинаковой четности не могут оказаться друг на друге.

Вы должны заметить, что

— диск с номером 1 перемещается один раз за любые два хода,

— он перемещается циклически, причем всегда в одном направлении, а именно

либо 0 — 1 1 — 2 2 — 0...
либо 0 — 2 2 — 1 1 — 0...

Следующий ход, перемещающий диск с номером 1, полностью определен. Недостаточно проверить это, это нужно доказать. После этого итеративное решение тривиально. Можете ли вы априори определить перемещение диска с номером 1 в зависимости от четности числа дисков?

Можете ли вы сказать что-нибудь о движении остальных дисков?

Пронумеруйте ходы. Диск с номером 1 перемещается в ходах с нечетными номерами. Проверьте, а затем докажите, что диск с номером 2 перемещается в ходах с номерами 2, 6, 10, . . ., т. е. в ходах, номер которых кратен двум, но не кратен четырем. Обобщите. Исходя отсюда, вы можете сказать, зная номер хода, какой диск будет перемещаться, с какого стержня он уйдет и куда придет.

Красиво, не правда ли?

Игра 34.

Существование четвертого стержня не упрощает стратегию, даже наоборот. Одна из возможностей состоит в том, чтобы перемещать p верхних дисков, используя 4 стержня. затем оставшиеся диски — используя только 3 стержня (поскольку четвертый стержень блокирован башней самых маленьких дисков). Наконец, вы восстанавливаете p маленьких дисков над остальными, используя 4 стержня. Обозначим через

$f4\ (p)$ — число ходов для перемещения p дисков, используя 4 стержня;

$f3\ (p)$ — число ходов для перемещения p дисков, используя 3 стержня (известное число, см. игру 31).

Тогда наша стратегия дает

$$f4\ (n) = f4\ (p) + f3\ (n - p) + f4\ (p).$$

Нужно выбрать значение p, которое минимизирует эту сумму.

Первые несколько значений для $f4$ получить легко:

$$f4(1) = 1,\ f4(2) = 3,\ f4(3) = 5.$$

В этих случаях на самом деле есть только один способ действовать. Вычислите сначала на руках следующие значения. Воспользуйтесь вашим компьютером, чтобы составить таблицу, дающую последовательные значения для $f4(n)$, вместе с оптимальным значением p для каждого n (оно не всегда однозначно определено. Вы по своему произволу можете выбирать из них наименьшее).

Игра 35.

Итеративная программа для игры с 4 стержнями есть обобщение итеративной программы для игры с 3 стержнями. Это видно по рекурсивной форме. Она не идеально проста...

Это замечание позволит вам перейти к любому числу стержней.

Игра 36.

Нужно снова взять все, что было нами оставлено в игре 23. Предположите, что для некоторого p существует такое значение q, что

$$SG(p,\ q) = 0.$$

Покажите, что в этом случае $SG(p,\ q') = 0$ для всех $q' < q$. Следовательно, если p таково, что $SG(p,\ 1) = 0$, то должно существовать некоторое g такое, что $SG(p,\ g) = 0$, но $SG(p,\ g + 1) \neq 0$; g — наибольшее из значений q, дающих равенство $SG(p,\ q) = 0$.

Нужно построить последовательность p_i, g_i.

Вы можете показать, что если $g_i = 1$, то $p_{i+1} = p_i + 2$, в то время как если $g_i > 1$, то $p_{i+1} = p_i + 3$.

Хороший способ действия состоит в том, чтобы опереться на геометрические рассмотрения. Числа Спрага—Грюнди интересуют нас только с одной стороны— равны они нулю или нет (у нас нет намерения играть несколько игр одновременно, что избавляет нас от вычисления Ним-сумм и, следовательно, от заботы о значениях ненулевых чисел Спрага—Грюнди). Число Спрага—Грюнди равно нулю тогда и только тогда, когда невозможен никакой переход к нулевому числу. Но положение p, q допускает переходы к $p - k$, для $k \leqslant 2q$. Следовательно, мы получим $SG(p,\ q) = 0$ тогда и только тогда, когда

$$SG(p - k,\ k) \neq 0 \text{ для всех } k \text{ от } 1 \text{ до } 2q.$$

Нарисуйте на плоскости две перпендикулярные оси, p как абсциссу и q как ординату. Обозначьте точки с нулевыми значениями SG.

Рассмотрите те прямые, которые проходят через точки p с SG $(p, 1) = 0$. Нужно изучить прямые $p - k$, k, где k меняется от 1, т. е. те, которые параллельны биссектрисе

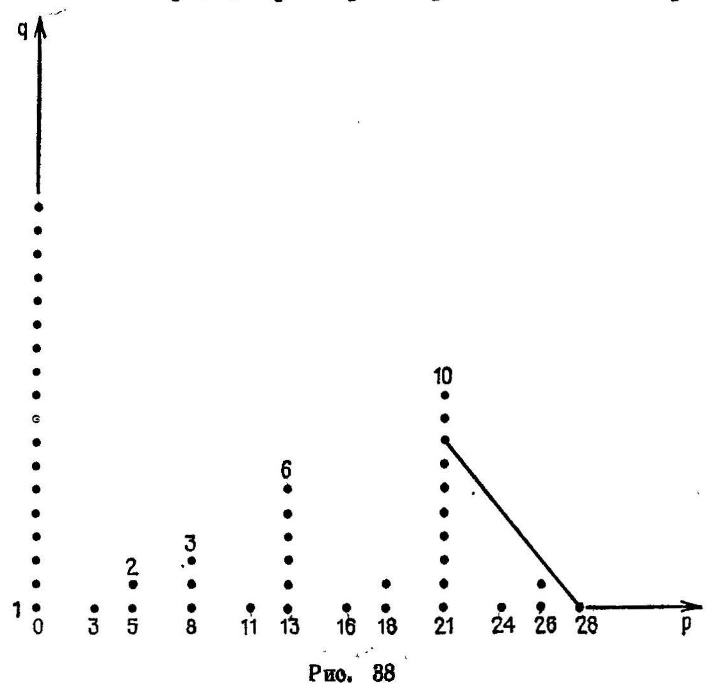


Рис. 38

второго и четвертого координатного угла и проходят через точку $p - 1$, 1.

Мы представили отрезок такой прямой для $p = 28$ (см. рис. 38). Он пересекает точку с нулевым значением на вертикали $21 = 28 - 7$. Значит, нужно ограничить число k шестью, задавая $g = 3$ при $p = 28$.

Для $p = 34$ диагональ, проходящая через 33, 1 проходит над всеми отрезками с 0 для $p \neq 0$ и пройдет поэтому, пересекая ось q при $q = 34$. Поэтому нужно ограничить число k тридцатью тремя и, следовательно, взять $g = 33 : 2 = 16$.

У вас есть также некоторое число таких p_i, что диагональ, выходящая из $p_i - 1$, 1, не пересекает никакого отрезка нулей перед осью q, что дает $g_i = (p_i - 1) : 2$.

Исходя отсюда, следующие числа p определяются диагоналями, которые перерезают вертикальный отрезок, выходящий из p_i, так, что $p - p_i \leqslant g_i = (p_i - 1) : 2$. Тогда можно восстановить первоначальную последовательность, несущую нули, вплоть до $(p_i - 1) : 2$.

Теперь вы легко сможете доказать, что интересующая нас последовательность p_i есть последовательность чисел Фибоначчи.

Составьте программу, перечисляющую p_i, g_i.

6. КОМБИНАТОРНЫЕ ЗАДАЧИ

Головоломка 20. Полное решение.

Поскольку эта задача всюду решена, предложим также и здесь решение: это избавит вас от поисков других решений; и, кроме того, я буду уверен, что вы посмотрели на все существенные места этой задачи. Есть книги, которые... Но это — совсем другая история.

Заметим сначала, что два ферзя не могут находиться на одной строке (горизонтали) и, поскольку нужно поставить 8 ферзей на 8 строк, то на каждой строке есть ферзь. Поэтому я буду говорить «ферзь k» вместо «ферзь, стоящий на строке k».

Точно также, есть только один ферзь в каждом столбце. Но совершенно ясно, что я не могу управлять в одно и то же время размещением и по строкам и по столбцам — собственно, это от меня в задаче и требуется. Я собираюсь поэтому размещать ферзей на последовательных строках, начиная сверху.

Чтобы начать, я помещаю ферзя в первый столбец на первой строке. Тогда мне остается решить меньшую задачу: разместить 7 ферзей на 7 последних строках шахматной доски, учитывая, что ферзь стоит на первом поле первой строки. Я получу тогда все решения с ферзем 1 в столбце 1. Затем я поставлю ферзя 1 в столбец 2 и разрешу задачу с 7 ферзями, и т. д. — 8 раз.

Обобщим. Мы собираемся решить частную, но нужную задачу: полагая, что уже есть ферзи, правильно размещенные на строках от 1 до $k - 1$, и зная их положение, найти все возможные решения, размещая подходящим образом ферзей с номерами от k до 8. Обозначим программу, которая это делает, через HR (k)*). Стратегия очень проста:

*) Маленькая головоломка для знающих французский (или хотя бы имеющих словарь): откуда это обозначение?— *Примеч. ред.*

— мы пробегаем все поля на строке k,

— если поле свободно (т. е. не бьется уже поставленными ранее ферзями), то мы ставим на него ферзя k и решаем ту же задачу для $k + 1$.

При $k = 8$ задача проще всего. Не может быть более одного свободного столбца. Если он есть, то мы ставим туда последнего ферзя и записываем полученное таким образом решение. Если свободного столбца нет, то нет и решения.

Для задачи HR (k) необходимо знание состояния игры, получающегося после размещения первых $k - 1$ ферзей. Это предполагает по крайней мере, что известны столбцы, занятые этими ферзями. Может быть, следовало бы сказать больше. Обозначим символически «занять k, i» операцию, которая констатирует факт, что в стоблце i на строке k помещен ферзь.

```
HR (k) =
   ДЛЯ i := 1 ДО 8 ВЫПОЛНЯТЬ
     ЕСЛИ место k, i свободно ТО
         занять k, i
       ЕСЛИ k = 8 ТО выписать решение
           ИНАЧЕ HR (k + 1)
       КОНЕЦ_ЕСЛИ
       освободить k, i
     КОНЕЦ_ЕСЛИ
ВЕРНУТЬСЯ
```

Операция «освободить k, i» отменяет то, что делает операция «занять k, i». Для решения задачи нужно изложить последовательность инициализации, отмечающую, что ничего не сделано и ни один ферзь в игре не участвует, а затем вызвать HR (1).

Эта процедура рекурсивна, так как она обращается сама к себе. Тщательно изучите ее. Если вы исходите из гипотезы, что HR ($k + 1$) находит и выводит все такие решения, у которых первые k ферзей стоят там, где они поставлены, то у вас не будет никаких затруднений в том, чтобы убедиться, что эта процедура совершенно правильна. Используйте крайние случаи: $k = 8$ и начальное обращение с $k = 1$.

Если у вас в наличии нет никакого другого языка, кроме Бейсика, или если вы раб своего языка до такой степени, что не желаете учить что-нибудь, кроме Бейсика, то вам придется писать итеративное решение. Это сложнее.

Будем исходить из наиболее общей ситуации. Пусть на шахматной доске уже размещено $k - 1$ ферзей. Обозна-

чим это состояние буквой С (в смысле «самое общее состояние»). Это состояние раскладывается на три подсостояния:

— уже размещено по местам 8 ферзей ($k - 1 = 8$): состояние С8;

— на строке с номером k есть допустимое место для ферзя: состояние СОК;

— либо строка с номером k блокирована полностью, либо все возможные поля на ней уже исследованы: СБ.

Запишем кусок программы, который различает эти три случая:

```
С: ЕСЛИ k=9 ТО С8
   ИНАЧЕ искать первое свободное поле
     на строке k и придать значение этого
     поля величине i;
     ЕСЛИ нет таких полей ТО СБ
     ИНАЧЕ СОК КОНЕЦ_ЕСЛИ
КОНЕЦ_ЕСЛИ
```

Рассмотрим теперь каждое из подсостояний.

СОК: есть свободное место в точке k, i. Туда ставим ферзя k и получаем снова самое общее состояние с еще одним размещенным ферзем.

Формально:

```
СОК: занять k, i; k := k + 1; С
```

Если строка k блокирована, а также если она полностью исследована, то нужно изменить выбор, который был сделан для ферзя $k - 1$, и передвинуть его на свободное место правее (если оно есть). Это возвращение назад относится непосредственно к ферзю $k - 1$ и, следовательно, сохраняет только $k - 2$ первых ферзей, что вызывает необходимость уменьшить k на 1. Может случиться, что это приведет нас к $k = 0$, т. е. может случиться, что все места на строке 1 уже исследованы и, следовательно, работа закончена, что мы обозначим как состояние Я, конец программы.

```
СБ: k := k − 1;
   ЕСЛИ k = 0 ТО Я
     ИНАЧЕ найти место i ферзя k;
       освободить k, i;
       найти первое свободное поле на строке k, рас-
       положенное правее i, и придать значение этого
       поля величине i;
       ЕСЛИ нет таких полей ТО СБ
       ИНАЧЕ СОК КОНЕЦ_ЕСЛИ
КОНЕЦ_ЕСЛИ
```

Когда 8 ферзей уже размещены, нужно записывать решение. Бесполезно искать другое место для восьмого ферзя, потому что если на восьмой строке и есть свободное место, то только одно. Таким образом, строка 8 оказывается полностью исследованной и нужно снова размещать 7 предыдущих ферзей. А состояние, в котором строка 8 полностью исследована,— это состояние СБ с $k = 8$.

```
С8: выписать решение;
    найти место i ферзя 8;
    освободить 8, i;
    k := 8; СБ
```

Остается пустить этот процесс в ход. В начале ни один ферзь в игре не участвует и, следовательно, $k - 1 = 0$. Нужна инициализация, которая бы это открыто провозглашала:

```
ПРОГРАММА: k := 1; инициализировать игру; С
```

Объединим куски. Мы получим программу, реализующую автомат, как мы уже видели в игре 12. Вы можете рассматривать имена, написанные прописными буквами (С, СБ, СОК, С8, ПРОГРАММА) как метки, позволяющие отсылать к части программы, в начале которой стоят эти имена со знаком «:» после них, и как инструкцию ПЕРЕЙТИ К, если они указаны в конце последовательности операций. Поэтому все это непосредственно переводится на совершенно любой язык.

```
ПРОГРАММА: k := 1; инизиализировать игру; С
С: ЕСЛИ k = 9 ТО С8
     ИНАЧЕ искать первое свободное поле на строке
       k и придать значение этого поля величине i;
       ЕСЛИ нет таких полей ТО СБ
       ИНАЧЕ СОК КОНЕЦ_ЕСЛИ
КОНЕЦ_ЕСЛИ
СОК: занять k, i; k := k + 1; С
СБ: k := k − 1;
     ЕСЛИ k = 0 ТО Я
       ИНАЧЕ найти место i ферзя k;
         освободить k, i;
         ИСКАТЬ первое свободное поле на строке
         k, расположенное правее i, и придать значение
         этого поля величине i;
         ЕСЛИ нет таких полей ТО СБ
         ИНАЧЕ СОК КОНЕЦ_ЕСЛИ
     КОНЕЦ_ЕСЛИ
```

C8: выписать решение;
найти место i ферзя 8;
освободить 8, i;
$k := 8$; СБ

Мы можем улучшить эту программу. Неприятно иметь необходимость находить заново место ферзя в строке, тем более, что знание этого места необходимо для вывода на экран полученного решения. Заменим i номером $c[k]$ столбца, где расположен ферзь k. Тогда искать место этого ферзя больше не нужно. Именно операция «занять k, i» и будет давать величине $c[k]$ значение i.

У нас есть два похожих отрывка в программе:

— в СБ:

искать первое свободное поле на строке k, расположенное правее i, и придать значение этого поля величине i;
ЕСЛИ таких полей нет ТО СБ
ИНАЧЕ СОК КОНЕЦ_ЕСЛИ

— в С:

искать первое свободное поле на строке k и придать значение этого поля величине i;
ЕСЛИ таких полей нет ТО СБ
ИНАЧЕ СОК КОНЕЦ_ЕСЛИ

Второй отрывок идентичен первому, если вместо того, чтобы искать первое свободное поле (что подразумевается как начальный ход), мы потребуем искать первое свободное поле после i, где i придано значение 0. Эту общую последовательность команд мы назовем И (от «искать»). Вот новая программа:

ПРОГРАММА: $k := 1$; инициализировать игру; С
С: ЕСЛИ $k = 9$ ТО С8
ИНАЧЕ $c[k] := 0$; И
КОНЕЦ_ЕСЛИ
И: искать первое свободное поле на строке k после $c[k]$ и придать значение этого поля величине $c[k]$;
ЕСЛИ таких полей нет ТО СБ
ИНАЧЕ СОК КОНЕЦ_ЕСЛИ
СОК: занять k, $c[k]$; $k := k + 1$; С
СБ: $k := k - 1$;
ЕСЛИ $k = 0$ ТО Я
ИНАЧЕ освободить k, $c[k]$
И

```
    КОНЕЦ_ЕСЛИ
  С8: записать решение;
    k := 8; освободить k, c [k], СБ
```

Мы можем еще немного выиграть. Значение 9 для k не может быть достигнуто иначе как после размещения ферзя на строке 8 с помощью СОК. Вместо того, чтобы проверять справедливость соотношения $k = 9$ в С, можно сделать это в СОК. Если нужно разместить восьмого ферзя, то бесполезно требовать «занять k, t» с тем, чтобы сразу после этого освободить указанное поле. Отсюда — новая, еще более простая программа.

```
  ПРОГРАММА: k := 1, инициализировать игру; С
  С: c [k] := 0; И
  И: искать первое свободное поле на строке k после
    c [k] и придать значение этого поля величине c [k];
    ЕСЛИ таких полей нет ТО СБ
    ИНАЧЕ СОК КОНЕЦ_ЕСЛИ
  СОК: ЕСЛИ k = 8 ТО записать решение; СВ
      ИНАЧЕ занять k, c [k]; k := k + 1; С
    КОНЕЦ_ЕСЛИ
  СБ: k := k — 1
    ЕСЛИ k = 0 ТО Я
      ИНАЧЕ освободить k, c [k]; И
    КОНЕЦ_ЕСЛИ
```

Дальше можно выиграть не так уж много, и мы в своих преобразованиях, направленных на улучшение программы, остановимся здесь. Читатель мог бы и удивиться моему способу работать: почему нельзя сразу дать хорошую программу? Потому что, по моему мнению, ее трудно получить сразу. Я мог бы с помощью мелких замечаний представить ее вам без каких-либо промежуточных рассуждений. Читатель был бы восхищен моей сноровкой, но, может быть, заявил бы, что программы такого рода ему самому недоступны, и отказался бы и от этого упражнения, и от остальных упражнений из этого семейства. Если, напротив, читатель находит последнюю программу очевидной, то это потому, что его интуиция намного богаче моей, и он выходит из этой работы ободренный: он еще более ловок, чем автор, браво! И во всех случаях я выигрываю.

Перечитаем нашу программу, чтобы лучше понять ее стратегию. Мы начинаем с пустой шахматной доски. Строчка за строчкой мы ищем первое свободное поле и занимаем

его. Это — цикл, который идет от С к И, затем в СОК и затем в С, и который останавливается, когда либо все ферзи уже размещены (выход в СБ из СОК), либо, что более вероятно, когда одна из строк блокирована (выход в СБ из И).

Если строка блокирована (или после того, как решение выписано), мы поднимаемся строчкой вверх ($k := k - 1$ в СБ), освобождая ферзей, пока не окажется возможным передвинуть какого-то ферзя правее (цикл СБ, И, СБ из И). Как только оказывается возможным переместить ферзя правее, он туда перемещается и возобновляется спуск.

Учитывая все это, мы видим, что наша стратегия достаточно проста и выглядит естественной, как только мы к ней привыкаем: ведь привычка — вторая натура, не так ли?

Существенное замечание: я говорю о программе так, как будто она закончена. Но еще ничего завершенного нет: вы никак не можете ввести эту программу в машину, потому что все записано символически. Как вы узнаете, является ли поле свободным? Что это такое — занять поле? Такая ситуация не является исключительной: мы можем обсуждать стратегию программы, совсем не обсуждая представление данных. Две вещи полностью разделены:

— алгоритм или стратегия, которой мы следуем при проведении вычислений;

— структуры данных, или способ представления элементов вычислений посредством основных типов, имеющихся в распоряжении используемого языка (в основном: числа, символы, таблицы или массивы чисел, цепочки символов).

Это — один из фундаментальных принципов программирования: стараться отложить на как можно более позднее время любое решение относительно выбора наиболее удобного представления данных. Рассмотрите сначала стратегию, которой вы следуете, используя символические формулы, которые вы впоследствии разовьете. Есть только две возможности:

— либо, как в рассматриваемом случае, вы приходите к цели. Как только этот первый этап пройден, вы спокойно обсуждаете представление данных;

— либо вы не в состоянии добраться до цели вследствие некоторого влияния структуры данных на стратегию. Такое бывает. Когда вы не можете продвинуться дальше в разработке стратегии, тогда начинайте с выбора представления данных, в котором вам послужит все то, что вы уже сделали к этому времени, и вы учтете то, что вас остановило.

Программирование всегда должно идти нисходящим путем. Сначала — алгоритм или стратегия. Потом — структура данных.

Посмотрим, какие структуры данных возможны в нашей задаче. Первая, наиболее естественная идея: я представляю шахматную доску с помощью квадратной таблицы с 8 строками и 8 столбцами. Я ставлю нули на пустые клетки. Чтобы найти свободное поле на строке, я перебираю поле за полем на строке, пока не нахожу поле с нулем. Это просто. Но как теперь занять поле k, c [k]? Поместив туда значение k. Это тоже просто. Но ферзь, которого нужно разместить, бьет некоторое количество полей, и их уже нельзя будет в дальнейшем занимать. Чтобы это учесть, нужно записать значение k по всем ранее свободным полям, которые теперь бьет этот новый ферзь. Здесь нужен цикл для занятия полей под ферзем на той же вертикали, а затем два других цикла — для каждой из диагоналей, проходящих через это поле (бесполезно занимать поля строки, потому что строка больше рассматриваться не будет). Это проще всего. Что касается освобождения, то нужно пробежать по шахматной доске и заменить там все значения k нулями. Очень долго...

Но как же иначе? Если что и составляет существенную необходимость, то именно знание, можно использовать поле или нет. Как бы я поступил при работе вручную? Выяснил бы, есть ли ферзи в том же столбце или на диагоналях, проходящих через это поле. Следовательно, мне достаточно знать состояние занятости столбцов и диагоналей. Я могу найти выход с помощью трех таблиц: одна — для столбцов, другая — для левых диагоналей, третья — для правых диагоналей. Чтобы узнать, свободно ли поле, я стану выяснять, свободны ли проходящие через него диагонали и столбец. Чтобы занять поле, я отмечу, что его столбец и диагонали заняты. Чтобы его освободить, я отмечу, что они свободны. Циклов больше нет. Вот хорошее решение.

Таким образом, нужен вектор с 8 полями, чтобы сказать, свободны ли столбцы. Обозначим этот вектор *ст*. Тогда *ст* [i] $= 0$ будет означать, что в столбце i нет ни одного ферзя. Его не надо путать с c [k], который отвечает на вопрос, в каком столбце стоит ферзь k.

Диагонали характеризуются тем условием, что сумма или разность номеров строки и столбца постоянны. Обозначим через *дп* диагонали, соответствующие сумме, *дм* — диагонали, соответствующие разности. В первом прибли-

жении диагонали, соответствующие полю k, i, суть $\partial п\,[k+i]$ и $\partial м\,[k-i]$.

Но при $1 \leqslant k \leqslant 8$, $1 \leqslant i \leqslant 8$ сумма меняется от 2 до 16, а разность — от −7 до 7. Чтобы остаться в промежутке от 1 до 15 (чего некоторые языки просто требуют), нужно вычитать 1 из суммы и прибавлять 8 к разности. Тогда диагонали, проходящие через k, i, суть $\partial п\,[k+i-1]$ и $\partial м\,[k-i+8]$.

Операция «искать первое свободное поле...» реализуется маленьким циклом в программе. Вот — на псевдоязыке, используемом в этой книге и близком к Бейсику, LSE и языку Паскаль,— что из всего этого получается:

```
    ТАБЛИЦА с[8], ст[8], дп[15], дм[15]
        k := 1
    ДЛЯ j := 1 ДО 8 ВЫПОЛНЯТЬ
        ст [j] := 0
    ВЕРНУТЬСЯ
    ДЛЯ j := 1 ДО 15 ВЫПОЛНЯТЬ
        дп [j] := 0; дм := 0
    ВЕРНУТЬСЯ
С   с [k] := 0
И   i := с [k] + 1
    ВЫПОЛНЯТЬ
        ЕСЛИ i = 9 ТО КОНЧЕНО
        КОНЕЦ_ЕСЛИ
        ЕСЛИ ст [i] = 0 И дп [k + i − 1] = 0 И
          дм [k − i + 8] = 0 ТО КОНЧЕНО
        КОНЕЦ_ЕСЛИ
        i := i + 1
    ВЕРНУТЬСЯ
    ЕСЛИ i = 9 ТО ПЕРЕЙТИ К СБ КОНЕЦ_ЕСЛИ
СОК  с [k] := i
    ЕСЛИ k = 8 ТО ВЫВЕСТИ с;
    ПЕРЕЙТИ К СБ КОНЕЦ_ЕСЛИ
    ст [i] := k; дп [k + i − 1] := k;
    дм [k − i + 8] := k; k := k + 1
        ПЕРЕЙТИ К С
СБ  k := k − 1
    ЕСЛИ k = 0 ТО ПЕРЕЙТИ К Я КОНЕЦ_ЕСЛИ
    i := с [k]; ст [i] := 0; дп [k + i − 1] := 0;
    дм [k − i + 8] := 0
    ПЕРЕЙТИ К И
Я   КОНЕЦ_РАБОТЫ
```

У вас теперь есть все, что только может быть вам нужно для того, чтобы это заработало на вашем компьютере.

Что касается симметрий, то вот указание. Эта программа заставляет первого ферзя пробежать всю первую строку. Но достаточно, чтобы он пробежал половину, а затем дополнить результат по симметрии. Остановить пробег, когда с [1] достигает значения 4, нелегко, но легко начать пробег с цифры 5. Ну, уж теперь-то я сказал вам достаточно...

Я не знаю простого решения для симметрии относительно диагонали. Если вы найдете такое решение, напишите мне...

Головоломка 21.

Я не вижу способа взяться за эту задачу, существенно отличного от предыдущего. Нужно найти нижнюю границу для числа ферзей. На пустой шахматной доске ферзь может блокировать 28 полей. Следовательно, нужно по крайней мере 3 ферзя, чтобы блокировать доску. Их нужно не больше 7: если вы уже пытались вручную поставить 8 ферзей, то вы должны были убедиться, что шахматная доска часто блокируется до того, как мы смогли поставить восьмого ферзя. Точно так же вероятно, что 6 ферзей должно хватить. Поэтому нужно исследовать отрезок от 3 до 6 ферзей.

Нет никакой уверенности в том, что эти ферзи не должны бить друг друга. Конечно, на шахматной доске есть поля, которые бьются по крайней мере двумя ферзями. Но нужно иметь возможность ограничить поиск решениями, для которых никакие два ферзя не бьют друг друга, или, может быть, немного проще — решениями, для которых никакие два ферзя не стоят на одной строке.

Вы размещаете k ферзей. Вы пробегаете шахматную доску в поисках свободного поля. Если его нет, то у вас есть решение. Если свободное поле есть, то вы ставите туда ферзя и начинаете поиск сначала. Бесполезно пробегать строки, на которых ферзь уже есть. Это соображение ускоряет проверку.

Головоломка 22.

Ничего трудного. Нужно перепробовать все комбинации. Берем какую-нибудь шашку домино в качестве начальной шашки цепочки и пробуем шашки одну за другой. Они вынимаются из хранилища, а затем отыскивается первая шашка, которую можно связать с данной, тем же способом, которым отыскивалось первое свободное поле на следующей строке.

Тщательно выберите ваше представление шашек домино.

Головоломка 23.

И на этот раз программирование достаточно просто. Вы задаете крайние члены последовательности:

$a_1 = 0,\ a_n = k.$

С помощью уже проведенного рассуждения вы можете зафиксировать

$a_2 = 1,\ a_{n-1} = k - 2.$

Затем вы размещаете следующие члены в интервале $(2,\ k - 3)$, например, уплотняя их к началу:

$a_3 = 2,\ a_4 = 3,\ a_5 = 4\ldots$

Вы образуете разности и, если они дают слишком много повторений (вы можете узнать его, не вычисляя всех разностей, что ускоряет тест), вы увеличиваете последний подвижный член a_{n-2} и, когда добираетесь до конца, увеличиваете предпоследний подвижный член, затем берете $a_{n-2} = a_{n-3} + 1$ и продолжаете дальше.

Для последовательности с 5 членами есть только один подлежащий размещению член, и все идет очень быстро. Но сложность растет с ростом n очень круто. Если при 5 членах есть только один подлежащий размещению член, то с $n = 6$ их уже два и задача квадратична. Для произвольного n число подлежащих испытанию случаев имеет порядок n^{n-4}.

Можно, наверное, и еще ускорить. Если даны n и k (значение последнего члена), то известно максимальное число возможных повторений, и можно выбрать наилучшие исходные значения. Если есть право на r повторений, то можно брать не более $r - 1$ последовательных членов, начиная с a_2, и, если они взяты как исходные значения, то права на повторение больше нет. Тем не менее эта задача расходует огромное количество машинного времени...

Головоломка 24.

В этой задаче я вас полностью предоставляю себе. Принцип все тот же. Но нужно как следует все организовать. Желаю успеха.

Головоломка 25.

Здесь, наоборот, помощь может оказаться далеко не лишней. Эта программа потребовала от меня массу времени. Кроме того, это поучительный случай, который я сохраняю в своих архивах как типичный пример для целого класса задач.

Среди информатиков есть два принципиально разных взгляда на программирование. Есть школа, приверженцы

которой сначала проделывают всю математическую работу: они считают, что для того, чтобы написать хорошую программу, нужно сначала доказать некоторое свойство данных, а затем использовать его для получения результата. Сначала сделайте математику, а информатика придет позже. Таким образом, это способствует рассмотрению информатики как ветви математики.

Но есть и другой подход. Напишите сначала программу, пусть даже неэффективную. Затем понаблюдайте за ее поведением или постарайтесь прояснить ее действие. С помощью подходящих преобразований сделайте ее более результативной. Довольно часто я получаю таким образом весьма эффективные результаты, и я убежден, что в этом состоит новый метод создания алгоритмов. Но бывают упорно сопротивляющиеся случаи. Эта головоломка — один из них.

Начну со следующего замечания: речь идет о том, чтобы образовать все возможные перестановки и выбросить все те, которые не удовлетворяют условиям задачи.

Рассмотрим сначала случай 9 девушек. Обозначим их

а, б, в, г, д, е, ж, з, и.

Первая прогулка может быть выбрана произвольно. Возьмем:

а б в
г д е
ж з и

Беря в качестве строк столбцы этой таблицы первой прогулки, получаем вторую прогулку:

а г ж
б д з
в е и

Диагонали приводят к двум оставшимся прогулкам:

а д и　　*а е з*
г з в　　*б г и*
б е ж　　*в д ж*

Все благополучно. Попробуем теперь 15.
Первая прогулка

а б в
г д е
ж з и
к л м
н о п

Если вы возьмете в качестве трех первых строк второй прогулки начала столбцов первой прогулки:

а г ж
б д з
в е и

вы не сможете далее организовать 6 оставшихся букв в двух строчках, не повторяя пар. Но вы можете сохранить первые пары в этих трех строках и взять в качестве последних элементов соответственно *ж*, *к*, *н*. Посредством этого приема получаются в двух столбцах по три неиспользованных элемента, которые и можно взять в качестве новых строк. Получается вторая прогулка:

а г ж
б д к
в е н
з л о
и м п

Сейчас мы докажем некоторые свойства искомых прогулок. Но здесь я делаю вам подарок. Мне потребовалось несколько дней, чтобы сообразить все то, что следует ниже. Почему бы вам не предоставить себе несколько дней на размышление? Тогда закройте книгу на этом месте...

Рассмотрим подмножество из семи букв *а*, *б*, *в*, *г*, *д*, *е*, *ж*. Исходя из этих элементов, можно образовать $7 * 6/2 = 21$ пару. В первой прогулке участвует 6 из этих пар:

а — б а — в б — в з — д г — е д — е

Во второй прогулке их пять:

а — г а — ж г — ж б — д в — е

что составляет всего 11 пар. Таким образом, на оставшиеся 5 прогулок остается распределить 10 пар. Но поскольку есть 7 элементов и только 5 строк, то в каждой прогулке будет встречаться не менее двух таких пар. Следовательно, в каждой из оставшихся прогулок встретятся в точности две таких пары. Обозначим через x любую из выделенных букв, а остальные буквы будем обозначать точками. Оставшиеся 5 прогулок имеют вид

x x .
x x .
x . .
x . .
x . .

Но можно еще кое-что уточнить. Рассмотрим только первые 6 букв *а*, *б*, *в*, *г*, *д*, *е*. Они дают 15 пар, из которых 9 реализуются в двух первых прогулках. Таким образом, среди 5 оставшихся прогулок надлежит распределить 6 из них, что означает по одной паре в четырех из них и две в последней. Поэтому получаем:

```
х х .   х х .   х х .   х х .   х х .
х ж .   х ж .   х ж .   х ж .   х х .
х . .   х . .   х . .   х . .   х . .
х . .   х . .   х . .   х . .   х . .
х . .   х . .   х   .   х . .   ж . .
```

Заменим *ж* на *к* или *н* и получим тот же результат. Покажите самостоятельно, что в конце концов получаются следующие схемы

```
х х .   х х .   х х .   х х .   х х .
х ж .   х ж .   х ж .   х ж .   х х .
х к .   х к .   х к .   х к .   х . .
х н .   х н .   х н .   х н .   х . .
х . .   х . .   х . .   х . .   ж к н
```

Вам остается расставить сначала *а*, *б*, *в*, *г*, *д*, *е* вместо букв *х*, не возобновляя уже использованных пар, а затем расставить буквы *з*, *и*, *л*, *м*, *о*, *п* вместо точек, соблюдая то же правило. На моем компьютере это отнимает не более 3 минут.

Эффект впечатляющий. Здесь мы можем правильно оценить истинную природу комбинаторных задач. Они сложны — иначе говоря, они требуют много времени для вычислений (именно в этом смысле и употребляется слово «сложный» в информатике). Предварительное доказательство подходящих свойств позволяет избежать слишком большого числа попыток и, следовательно, уменьшить сложность. Остается только найти эти хорошие свойства...

Головоломка 26.

Пентамино является другим примером этого утверждения. Общая идея решения проста, если учесть все то, что вы уже сделали. Вы рассматриваете прямоугольную область, которая должна быть покрыта различными кусочками и в начале игры должна быть обозначена вами как пустая.

Вы можете действовать двумя способами:

— рассматриваете первое свободное поле и ищете кусок, который можно туда поместить;

— берете первый еще не использованный кусок и пытаетесь поместить его на игровое поле.

Кусок может быть по-разному ориентирован. Если «I» (прямой брус) может быть размещен в прямоугольнике 3 × 20 только одним способом (параллельно большей стороне), то «F» (вроде правой нижней фигуры на рис. 31) может быть ориентирован восемью способами. Это зависит в первую очередь от симметрии кусков.

Чтобы не было необходимости определять, какие ориентации допустимы, вы можете задать — в качестве программных констант — все эти возможные положения каждого куска.

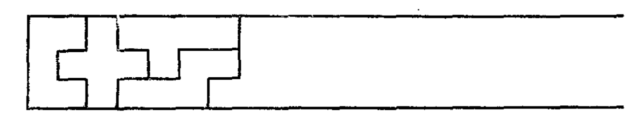


Рис. 39

Вы можете составить программу без каких-либо хитростей. Кажется, что более эффективно брать первое пустое поле и пытаться поместить туда какой-либо кусок. Вы ищете первое свободное поле. Вы рассматриваете первый еще не использованный кусок. Вы исследуете в некотором порядке все его ориентации, чтобы выяснить, приемлема ли какая-нибудь из них — покрывает ли она только свободные поля. Если игра блокирована (никакой кусок поместить нельзя), то вы удаляете последний размещенный кусок и продолжаете поиск, начиная со следующей ориентации того же куска. Я пробовал сделать так, и это слишком долго...

Тогда я стал пытаться избежать большого числа испытаний, исходя из замечания, сделанного при постановке задачи: кусок не должен определять в игре «островок» с площадью, не кратной пяти. Но определение островков нетривиально...

Я действую следующим образом. Я отыскиваю заполнение прямоугольника параллельно меньшей стороне. Рисунок 39 показывает возможную ситуацию в ходе выполнения этого плана.

Рассмотрим тогда конфигурацию, окружающую крайнее левое из свободных полей. Обозначив через «x» занятые поля и полагая свободные поля точками, мы получим не более 7 возможных случаев (если вы привыкли к дво-

ичной нумерации, то это покажется вам очевидным): см. рис. 40.

В крайней левой ситуации будем искать способ занять свободное поле на верхней строке. Но ни один из кусков ни в какой из их ориентаций не подходит. Вы не можете использовать ни крест, ни «F», ни «Z». Кусок «С» можно использовать только с большей стороной по вертикали...

Я закрепил за каждой конфигурацией список допустимых в ней кусков, и если такие куски есть, подробный

x •	x x	x •	x x	x •	x x	x •
x •	x •	x x	x x	x •	x •	x x
x •	x •	x •	x •	x x	x x	x x

Рис. 40

список их возможных ориентаций. Это существенно уменьшает число попыток. Еще оказывается, что время от времени появляются острова недопустимой площади, но они существуют только очень короткое время. Я узнал это, поскольку я выводил на экран состояние игры всякий раз, когда в игру входил новый кусок. Этот способ действия имеет много преимуществ:

— очень неудобно иметь программу, которая работает несколько десятков минут (порядка 45 на моем микрокомпьютере), а мы ничего не знаем о том, что в ней происходит. Это неудобно как собственно для работы, так и для того, чтобы сразу же задавать вопросы. А если, хотя бы это и было ошибкой набора, вдруг найдется бесконечный цикл...

— этот вывод позволяет видеть работу компьютера. Видно, как один за другим исследуются куски, как игровое поле более или менее наполняется (иногда вплоть до одиннадцати кусков. Если вы пытались решить эту головоломку вручную, отметили ли вы, какое впечатление производит нехватка одного куска? Однако это просто: если остается островок площади 5, то он обязательно имеет форму одного из игровых кусков...). Затем она почти полностью опустошается, и возобновляется заполнение...

Конечно, вывод на экран требует машинного времени и замедляет работу программы. Всегда будет время отказаться от вывода на экран и переделать процесс выполнения программы без вывода на экран, чтобы получить точное время решения задачи. Чтобы вывод был красивым,

нужно, чтобы рамка оставалась на экране неподвижной. Сделать это более или менее легко в зависимости от системы программирования, имеющейся в вашем распоряжении.

Для вывода на экран я не нашел хорошего рисунка, потому что у меня нет ни графического, ни полуграфического экрана — только алфавитно-цифровой. Каждому куску я сопоставил букву и вывожу куски на экран в виде подходящим образом расположенных пяти букв. Такой вывод показан на рис. 41.

```
C C X Y Y Y Y . . . . . . . . . . . . .
C X X X Y N N . . . . . . . . . . . . .
C C X N N N . . . . . . . . . . . . . .
```

Рис. 41

Я представляю игру внутренним образом в виде цепочки символов по двум причинам:

— используемый мною язык (LSE) в используемой мною версии является одним из наиболее эффективных языков для работы с цепочками символов. Это почти также быстро, как если использовать таблицы. Я могу очень быстро найти первое свободное место, я могу очень быстро узнать, свободно ли поле (является ли символ на этом месте в цепочке точкой?);

— вывод мгновенный: я вывожу на экран три подцепочки на трех последовательных строках.

Остается еще установить немало деталей, касающихся представления кусков. Но вы же не ждете, что я за вас сразу и программу напишу?

Головоломка 27.

А эта программа простая. Вам нужно образовать выражение вида

$$a_1 \circ a_2 \circ a_3 \circ \ldots \circ a_p,$$

где операция, обозначенная $\circ$, означает либо сложение, либо вычитание. Есть $p-1$ знак, каждый из которых может принимать два значения. Это дает 2^{p-1} возможных значений. Каким бы ни был способ, которым вы их перебираете, вам нужно перепробовать их все (по крайней мере в случае, когда число s таково, что решения нет).

Два знака «+» и «—», так что снова двоичная система. Вы можете воспользоваться этим замечанием при составлении программы. Меняем целое число от нуля до

$2^{p-1} - 1$. Для каждого из значений рассматриваем его двоичное представление. Ставим в выражении «+» на тех местах, где стоят нули, и «—» на местах, где стоят единицы. Но в этом таится опасность побудить некоторых написать программу на языке ассемблера, что было бы ошибкой (по моему мнению. Вы тогда сплутовали бы. Есть хорошие алгоритмы на развитом языке. Не меняйте условий задачи, выписывая алгоритм, который оказался бы необъяснимым).

Вы можете также — и это, конечно, более эффективный способ — поставить знаки «+» в начале выражения и исчерпать все комбинации с тем, что осталось, затем заменить последний знак «+» на «—» и т. д. С четырьмя числами вы получите последовательно:

+++ ++— +—+ +—— —++
—+— ——+ ———

Состояние знаков хранится в таблице или в цепочке.

Заметьте, что рассматриваемая задача имеет простое рекурсивное решение. Достаточно испробовать две комбинации:

a_1+ — любая комбинация, которая может быть составлена из $p-1$ оставшихся шашек,

a_1- — любая комбинация, которую можно составить из оставшихся шашек.

Должно получиться:

$s = a_1 +$ — комбинация из $n-1$ чисел

или

$s = a_1 -$ — комбинация из $n-1$ чисел.

Заметим, что разность нужно брать по абсолютному значению.

Таким образом, остается искать способ представления $s + a_1$ или $s - a_1$ с помощью $n-1$ оставшихся шашек. Такую процедуру легко написать. Таблица чисел может быть глобальной величиной. Чтобы сохранять только $n-1$ чисел, кроме первого, достаточно сказать, что таблица рассматривается, начиная с индеса 2. Следовательно, нужна процедура, в которой в качестве параметров берутся:

индекс, начиная с которого должны рассматриваться числа,

сумма, которую нужно найти.

Итеративные формы программы, которые вы сможете написать, суть немедленные переводы на итеративный язык этой рекурсивной формы.

Головоломка 28.

Решение, набросок которого я привожу здесь, принадлежит не мне. Я нашел его вышедшим из-под пера Николь Брео Поликен и Оливера Герца в журнале «Персональный компьютер» (L'ordinateur individuel) за март 1983 г.

Однако я должен сознаться, что это решение меня глубоко поразило. Программа была действительно очень хорошо написана на языке Паскаль и с большим мастерством были использованы свойства вложения процедур, которые давали возможность формальные параметры или локальные переменные некоторые из этих процедур рассматривать как глобальные параметры для других процедур.

Я переписал это решение практически без изменений на LSE83 (намного более структурированная форма LSE, соединяющая преимущества структурирования языка Паскаль с возможностями манипуляций с цепочками символов, имеющихся в LSE, и, сверх того, облегчением программирования сверху вниз), и результат немедленно оказался удовлетворительным. Все это служит прославлению авторов. Как же могло тогда случиться, что пояснения, которые авторы дают к своей программе, до такой степени недоступны пониманию, что мне потребовался большой труд, чтобы достичь понимания их метода? Там, действительно, есть две или три «хитрости», которые гораздо больше заслуживали комментария, чем тот факт, что из-за рекурсии результаты записываются в порядке, обратном порядку их получения...

Сохраним предположения работы этих авторов, приведенные в условиях задачи. Зачем от них отказываться? Например, такая комбинация, как

$$n = p_1*p_2 + p_3*p_4 - p_5/p_6$$

не сводится ни к одной из предложенных форм.

Программа, написанная авторами, рекурсивна, но ее читателю доставляют затруднения две особенности ее написания:

— как я уже указывал, некоторые переменные, являющиеся локальными в одной процедуре, глобальны в другой... Конечно, это может быть обнаружено при внимательном чтении текста, но это и требует внимания;

— некоторые процедуры мультиформны и дают совершенно различные результаты в зависимости от значений формальных параметров.

Вернемся к задаче в той форме, в какой она была поставлена. Что нужно делать?

Сначала пройдем по таблице шашек от 1 до 6. Для каждой шашки p_i посмотрим, делится ли n на p_i. Если да, то нужно решать меньшую задачу: образовать число $n/p[i]$ с помощью пяти шашек, получаемых удалением шашки i из набора. Если n не делится ни на одну из шашек или если поиск шашки, на которую делится n, потерпел неудачу, то для каждой шашки i ищем решение задачи: образовать $n + p[i]$ или $n - p[i]$ с помощью 5 шашек, получаемых изъятием шашки i из набора. Но здесь мы довольствуемся решением, которое должно иметь вид произведения одной из шашек на комбинацию четырех остальных.

Цитируемые здесь авторы решают эту задачу изъятия некоторых шашек из набора переписыванием начальной таблицы шашек в другую, перепрыгивая через шашки, подлежащие изъятию.

Я действую по-другому. Я помещаю 6 шашек в таблицу из 6 чисел, скажем a. В начале они упорядочены и расположены в неубывающем порядке. Чтобы изънть шапку из этого множества, мне достаточно переставить ее с шестой шашкой, а затем работать с первыми 5 элементами таблицы a. Таким образом, я создаю две процедуры:

процедуру

П (p, x),

которая ищет способ представить x с помощью p первых значений таблицы a, причем это решение должно иметь вид произведения одной из шашек на некоторую комбинацию остальных (П поставлено для решения в виде Произведения);

процедуру

О (p, x),

которая ищет решения задачи о формировании x из p первых шашек, в котором результат имеет какую-нибудь из форм, предложенных в формулировке задачи (О — от Общее).

Программа довольствуется чтением 6 шашек (в порядке возрастания) и числа n, которое нужно найти, а затем вызывает О (6, n).

Вся задача состоит в том, чтобы поддерживать часть таблицы от 1 до p в неубывающем порядке. Это нетрудно. Вот схематическое описание процедуры П. В нем t является глобальной булевой переменной, которой присвоено начальное значение ЛОЖЬ.

```
П (p, x)
ЕСЛИ p < 3 ТО упрощенная форма;
КОНЧЕНО КОНЕЦ_ЕСЛИ
i := ›
ВЫПОЛНЯТЬ
   ЕСЛИ x = a [p] ТО t := ИСТИНА; КОНЧЕНО
   КОНЕЦ_ЕСЛИ
   up := x/a [p]; u = целая_часть(up)
   ЕСЛИ u = up ТО О (p — 1, u);
     ЕСЛИ t ТО ВЫВЕСТИ
       u, '*', a [p], '=', x
     КОНЧЕНО КОНЕЦ_ЕСЛИ
   КОНЕЦ_ЕСЛИ
   i := i — 1; ЕСЛИ i = 0 ТО
   КОНЧЕНО КОНЕЦ_ЕСЛИ
   переставить (i, p)
ВЕРНУТЬСЯ
```

Вы покажете, что часть от 1 до $p - 1$ остается расположенной в неубывающем порядке. Но при выходе из цикла в p стоит элемент, который меньше всех остальных. Следовательно, нужно восстановить исходный порядок в части от 1 до p, если t не принимает значения ИСТИНА (в противном случае все кончено). Это вы легко изобретете.

Процедура О вдохновляется той же идеей, но есть два цикла:

— один, приводящий в p все элементы один за другим;

— другой, который приводит в $p - 1$ элементы, расположенные ниже того, который попал в p.

В конце каждого цикла нужно восстанавливать порядок. Эти восстановления порядка могут показаться дорогостоящими. Они стоят не меньше переписывания одной таблицы в другую со сравнением каждый раз по трем индексам, где добавляются перестановки таблицы в качестве формальных параметров процедуры. Здесь a — глобальная таблица.

Наконец, нужно заметить, что эта процедура прекрасно подходит для итеративного переписывания. Создаем вектор x, дающий искомое число для каждого p. Как и выше, индексы i и j процедур П и О связаны с p. Наконец,

переменную p сделали глобальной. Мне кажется достаточно очевидным, что итеративная процедура не пойдет намного быстрее рекурсивной процедуры: придется делать много проверок, которые выполнялись автоматически на уровне машинного языка, исполняющей системой. Но это и есть способ выйти из положения в случае, если, к несчастью, у нас нет рекурсивности.

Если у вас есть предубеждения против рекурсии, то сейчас подходящий момент избавиться от них. И бросьте думать, что рекурсия всегда дорого обходится. Она всегда сокращает время программирования. Неверно, что она всегда приводит к более медленному вычислению (эта головоломка и есть пример). Я соглашусь с вами, что она всегда занимает немного больше места...

Эта процедура, действуя на 6 шашек

100 75 50 25 10 10,

быстро находит число 370, но терпит неудачу для 369.

7. ОБО ВСЕМ ПОНЕМНОГУ

Головоломка 29.

Эта задача также не должна была бы излагаться ошибающимис. людьми. Я пытался понять, где эти программисты оступаются. Я считаю, что есть две опасности:

— прежде всего нет никакой уверенности в том, что поступающее число удастся эффективно разместить между двумя числами таблицы. Оно может оказаться перед первым элементом и после последнего элемента. Так как эта возможность влечет появление некоторых особенностей, то наши программисты начинают с изучения этих случаев, что совершенно ненужно;

— далее поиск должен происходить с помощью разделения каждый раз таблицы на две части. Сравниваем x со средним элементом. Если он больше, то нужно искать его место в верхней полутаблице. В противном случае он — в нижней половине. Но средний элемент — это элемент с индексом $k = (1 + n)/2$ или, в наиболее общем случае, где рассматривается кусок таблицы, начинающийся в p и кончающийся в q, — элемент с индексом $(p + q)/2$. Конечно, рассматривается только целая часть дроби. По этой причине некоторые программисты опасаются, что это может заставить обращаться много раз к одному и тому же элементу, и тогда программа не остановится или может вызвать потерю элемента.

Это — пустые опасения. Возьмем как общую следующую ситуацию: пусть мы смогли найти такие два целых p и q, что

$a[p] < x \leqslant a[q]$, причем $p < q$.

Тогда все очевидным образом завершено, если $q = p + 1$.

В противном случае скачок между q и p не меньше 2, и так как p меньше q, то, следовательно, элемент с промежуточным номером

$r =$ целая_часть $((p + q)/2)$

обязательно отличается от элементов с номерами p и q, и вам нечего опасаться. Вы сравниваете x с элементом с индексом r и в зависимости от результата сравнения берете r либо как новую нижнюю границу p, либо новую верхнюю границу q.

Остается одна трудность. Как выбрать p и q, чтобы так пустить в ход процесс, чтобы выполнялось общее двойное неравенство? Всегда, когда приходится выполнять обращение к таблице, представляет интерес введение дополнительных элементов, освобождающих от влияния концов таблицы. Введем элемент с индексом 0, меньший, чем любой из тех x, к которым можно обратиться (мы отложим на более поздний срок решение вопроса, как мы можем сделать это эффективно), и элемент с номером $n + 1$, больший, чем все возможные x. Тогда x обязательно больше, чем $a[0]$, и меньше, чем $a[n + 1]$.

Тогда мы можем начать с $p = 0$ и $q = n + 1$. Напишите соответствующую программу, вовсе не заботясь заранее о значениях $a[0]$ и $a[n + 1]$ и оставляя в неопределенном положении задачу эффективного описания таблицы (некоторые языки, такие как Фортран или LSE, не допускают индекса ноль — один только бог знает почему...). Покажите, что единственный индекс, для которого фактически приходится читать значение элемента таблицы,— это индекс r. Так как r всегда строго содержится в интервале (p, q), причем p не убывает, а q не возрастает, то r всегда строго больше 0 и не меньше n. Таким образом, элементы 0 и $n + 1$ никогда не опрашиваются. Поэтому и нет необходимости их материализовывать. Объявите массив (таблицу) с индексом, пробегающим от 1 до n, и все пройдет без сучка и задоринки...

Головоломка 30.

Это — задача, на которой я заваливаю профессионалов. Совершенно очевидно, что обе цепочки символов ис-

рают одну и ту же роль. Следовательно, в программе есть симметрия, которая касается способа обращения с этими цепочками. Вот — более или менее символически — программа, которую пишут профессионалы:

```
100 i:= 0;  j := 0.
110 продвинуть i к ближайшему символу в цепочке
    a, не являющемуся пробелом
120 ЕСЛИ мы вышли из a ТО ПЕРЕЙТИ К 200
    КОНЕЦ_ЕСЛИ
130 продвинуть j к ближайшему символу в цепочке
    b, не являющемуся пробелом
140 ЕСЛИ мы вышли из b ТО ПЕРЕЙТИ К 300
    КОНЕЦ_ЕСЛИ
150 ЕСЛИ a [i] = b [j] ТО ПЕРЕЙТИ К 110
160 ПЕРЕЙТИ К 800
200 продвинуть j к ближайшему символу в цепочке,
    не являющемуся пробелом
210 ЕСЛИ мы вышли из b ТО ПЕРЕЙТИ К 900
    КОНЕЦ_ЕСЛИ
220 ПЕРЕЙТИ К 800
300 продвинуть i к ближайшему символу в цепочке a,
    не являющемуся пробелом
310 ЕСЛИ мы вышли из a ТО ПЕРЕЙТИ К 900
    КОНЕЦ_ЕСЛИ
800 результат := ЛОЖЬ; ПЕРЕЙТИ К 1000
900 результат := ИСТИНА
. . . . . . . . . . . . . . . . . . . . . . . . .
```

Эта программа понятна. В 150 находим два символа, не являющихся пробелами. Если они совпадают, то нужно продолжать маршрут, а если они различны, то и цепочки различны (строчка 800).

Если в 120 констатируется, что все символы цепочки a уже испытаны, причем каких-либо различий с уже изученными символами цепочки b не обнаружено, то имеется выбор одной из двух возможностей (строки 200 и 210):

— либо в цепочке b нет ни одного символа, не являющегося пробелом (что приводит к тому, что в поисках такого символа мы выходим из b), и цепочки совпадают (строчка 900), либо мы обнаруживаем в цепочке b символ, не являющийся пробелом; эта цепочка включает символы, не входящие в a, и, следовательно, результат есть ЛОЖЬ (строка 800).

То же самое происходит, когда исчерпывается цепочка b (из строчки 140 переход осуществляется к строчке 300).

Я попытаюсь сделать из этого головоломку. Еще не слишком поздно. Найдите ошибку и исправьте ее. Но вы можете составить намного лучшую программу.

Головоломка 31.

Вот несколько идей. Вы можете сначала «отсортировать» обе цепочки, переставляя символы в каждой из них, чтобы они оказались, например, в алфавитном порядке. Когда это сделано, то цепочки должны оказаться одинаковыми. Это очень тяжеловесно...

Вы можете взять первый символ первой цепочки и посмотреть, есть ли он во второй цепочке. Если ответ отрицателен, то цепочки не являются анаграммами друг друга. Если же ответ — «да», то изымите этот символ из второй цепочки и переходите ко второму символу в цепочке a. Это ведет, по вашему выбору, к рекурсивной или к итеративной процедуре. Внимание: если вы смогли полностью пробежать a и не нашли ни одного символа, не попавшего в b, проверьте, не осталось ли чего-нибудь в b...

Вы можете задать таблицу, имеющую столько же полей, сколько может быть различных символов в рассматриваемых цепочках. Если мы имеем дело с текстами и если пробелы считаются, то нужны 33 буквы и пустое место... Вы пробегаете первую цепочку и добавляете 1 в клетке, связанной с каждым встречаемым характером (вы считаете число случаев появления каждого знака). Затем вы пробегаете вторую цепочку и все пересчитываете (вычитая, а не складывая). Если в конце вы получаете таблицу, содержащую что-то кроме нулей, то цепочки не являются анаграммами.

Конечно, есть и другие способы действовать. Достоинства каждого из них зависят от обстоятельств. Для текста последний способ кажется достаточно хорошим, первый — явно плох.

Головоломка 32.

То что было только что сказано, остается приемлемым и в этой задаче. Но достоинства различных решений могут измениться. Продумайте их. Это позволит увидеть, что одна программа никогда не бывает абсолютно лучше другой. Это зависит от данных и часто еще и от используемого компьютера...

Головоломка 33.

Есть карточный пасьянс, который более или менее похож на эту задачу. Выберем из вектора его первый элемент и отложим его в сторону на запасное поле. Мы можем поместить на его место элемент $m + 1$, который и

должен перейти на поле 1. Теперь поле $m+1$ свободно. Туда можно перенести элемент, который должен его заполнить. Возьмем конкретный пример: $n = 10$ и $m = 4$. Элементы верхней части спускаются на 4 поля, а те, которые находятся в нижней части, поднимаются на 6 полей. Вот последовательные состояния вектора. Я не представляю здесь запасное поле, которое содержит элемент 1. Я помещаю в эти поля именно номера элементов в исходной конфигурации:

исходное положение:	1	2	3	4	5	6	7	8	9	10
		2	3	4	5	6	7	8	9	10
	5	2	3	4		6	7	8	9	10
	5	2	3	4	9	6	7	8		10
	5	2		4	9	6	7	8	3	10
	5	2	7	4	9	6		8	3	10

А теперь именно элемент 1 должен прийти на свободное поле, и этот цикл останавливается. Мы убеждаемся, что не все элементы перенесены. Все числа на нечетных местах уже находятся там, где должны находиться, а числа на четных местах не стронуты с мест. Но можно начать новый цикл той же длины, поместив 2 на запасное поле,— это завершит работу.

Предлагая эту задачу профессиональным программистам, я очень редко получал такое решение, потому что им не удавалось выяснить, что мы действительно перемещаем таким образом все элементы (в этом можно убедиться, подсчитывая число движений) и нет ли опасности дважды переместить один и тот же элемент, так что конечное состояние оказалось бы неправильным *).

Чтобы навести себя на правильный путь, заметьте, что если верхние элементы спускаются на m полей, то нижние элементы поднимаются на $n - m$ полей.

Вы не видите? Есть n полей. Вы работаете в арифметике по модулю n. По модулю n все элементы спускаются на m полей. На этот раз вы должны найти решение, учитывая влияние на ход решения наибольшего общего делителя m и n...**)

*) При чтении этого абзаца вспоминается шутка Маяковского. После одного из своих выступлений он получил из зала записку: «Мы с товарищем слушали ваши стихи и ничего не поняли». Маяковский ответил: «Нужно иметь умных товарищей».— *Примеч. ред.*

**) Если разрешить перемещать каждый элемент вектора не один раз, а два, то можно найти изящное и простое решение, в котором и речи нет ни о каких общих делителях. Представим себе элементы вектора расположенными по окружности на равных рас-

Головоломка 34.

Предположим, что мы уже проделали часть работы. Именно таким образом мы всегда должны начинать поиск решения. Мы ограничимся здесь случаем таблицы чисел, который предоставит нам полную возможность изучения различных стратегий и позволит вам записать несколько программ и сравнить их, не заставляя вас пускаться в манипуляции с более деликатными цепочками.

Следовательно, предположим, что мы прошли таблицу до номера i включительно. Пусть в пройденной части мы нашли, что элемент со значением x повторялся p раз, и пусть это — максимальное число повторений.

Но нужно еще более уточнить ситуацию в точке остановки. У нас есть две возможности.

— Первая идея: мы останавливаемся в конце равнинного участка.

Если $i = n$, то мы прошли всю таблицу, узнали наилучший равнинный участок, и все закончено. В противном случае мы пробегаем следующую равнину и измеряем ее длину r. Если $r \leqslant p$, то наилучшая равнина остается неизменной, а в противном случае именно последняя равнина и регистрируется заново как наилучшая, и мы возобновляем движение по таблице. Это просто, и это легко программировать. Запишите это решение, чтобы иметь возможность сравнить его с другими решениями.

— Вторая идея: мы останавливаемся в произвольной точке i. Мы оказываемся на некоторой равнине и уже нашли r элементов на этой равнине.

Если $i = n$, то проход таблицы завершен. Мы знаем наилучшую возможную равнину с p повторениями и равнину с r элементами на последнем проходе. Мы берем лучшую из этих двух, и все кончено.

В противном случае нужно продвинуться вперед на один элемент. Либо этот элемент равен непосредственно предшествующему элементу; мы все еще находимся на той же самой равнине, длина которой увеличивается. Либо он отличается от предыдущего; тогда оказывается пройденной равнина длины r, которую при $r > p$ нужно зарегистрировать как наилучшую. С другой стороны, нужно сказать, что новый элемент находится на равнине, которая в данный момент имеет длину 1.

стояниях друг от друга. Нам нужно повернуть эти векторы на угол. Используйте тот факт, что всякий поворот окружности можно получить, выполнив подряд две осевые симметрии. — *Примеч. ред.*

На первый взгляд, первая стратегия дает программу, содержащую два вложенных цикла: маленький внутренний цикл для прохода равнины и глобальный цикл для прохода всего вектора, равнина за равниной.

Вторая программа содержит только один цикл, пробегающий элементы вектора один за другим и в нужных местах исправляющий значение p. Следовательно, эта вторая программа лучше.

Но это не все. Не позволяйте обмануть себя видимостью. Обе эти программы пробегают вектор элемент за элементом. Если вы составляете вашу программу на Бейсике или LSE, используя операторы ПЕРЕЙТИ К, а не циклы ДЛЯ или FOR, то вы убедитесь, что эти два решения почти неотличимы, а второе решение требует двукратного написания теста, сравнивающего r и p, так что едва ли не чаще эта вторая программа оказывается хуже.

Но есть третья стратегия. Восстановим общую ситуацию: мы прошли часть вектора до номера i включительно и определили наилучшую равнину длины p с общим значением ее элементов, равным x. Точка остановки произвольна.

Известно, что нужно осуществить включение нового элемента. Поставим следующий вопрос: насколько этот новый элемент может изменить ситуацию? Ответ: если он оказывается принадлежащим равнине с длиной, большей p. Может ли он оказаться принадлежащим равнине с длиной, намного большей p? Нет, мы бы это уже заметили. Следовательно, новый элемент изменяет ситуацию, если он принадлежит равнине длины $p + 1$. Но такое может случиться, если он равен элементу, содержащемуся в p предыдущих полях.

В начале ничего не пройдено: $i = 0$, и нет ни одного повторения: $p = 0$.

```
i := 0; p := 0
ВЫПОЛНЯТЬ
    ЕСЛИ i = n ТО_КОНЧЕНО
    КОНЕЦ_ЕСЛИ
    i := i + 1
    ЕСЛИ a[i] = a[i - p] ТО x := a[i]; p := p + 1
    КОНЕЦ_ЕСЛИ
ВЕРНУТЬСЯ
```

Красиво, не правда ли?

Но можно сделать лучше. Тщательно рассмотрите эту программу. Вы должны суметь обнаружить, что мож-

но перескакивать через некоторое количество элементов без обращения к ним...

Головоломка 35.

Не позволяйте себе поддаться впечатлению от ограничений на сложность алгоритма. Вы не можете выделить все возрастающие подпоследовательности, чтобы найти лучшую из них, это было бы слишком длинно, и легко сделать что-нибудь попроще этого.

Воспользуемся снова той же самой техникой. Пусть мы прошли вектор вплоть до некоторой точки. Пусть мы получили соответствующие результаты, но, поскольку мы еще не знаем, в какой форме они нужны, мы оставим их на некоторое время неопределенными. В любом случае выглядит вероятным, что мы знаем наибольшую по длине возрастающую подпоследовательность пройденной части, без которой мы как будто лишены возможности добраться до конца вектора... Как и выше, поставим вопрос: насколько изменяет ситуацию появление нового элемента? Он может продолжить известную нам наиболее длинную последовательность, если он может быть поставлен в ее конец, и, следовательно, если он больше последнего элемента этой последовательности. А если это не так, то эту наиболее длинную подпоследовательность он изменить не может. Но он может продолжить более короткую подпоследовательность, которая постепенно может стать более длинной, если она медленнее растет.

Рассмотрим, например, последовательность

4 5 3 8 2 6 1 7

Если ограничиться тремя первыми элементами, то наиболее длинная возрастающая подпоследовательность — это

4 5

Добавим четвертый элемент, 8. Он может быть присоединен к концу этой подпоследовательности и дает возрастающую подпоследовательность длины 3:

4 5 8

Следующий элемент — 2 — ничего не меняет. Следующий — 6 — не может быть присоединен к концу последовательности длины 3, но он может быть присоединен к концу последовательности длины 2 — последовательности 4 5 — чтобы дать другую подпоследовательность

длины 3:

4 5 6

Эта последовательность меньше предыдущей, поскольку ее последний элемент меньше, и поэтому у нее больше шансов иметь возможность продолжаться. На самом деле, 7 может быть присоединено к ее концу, что дает максимальную возрастающую последовательность

4 5 6 7

Мы уже видим, что нужно уточнить понятие максимальной возрастающей подпоследовательности, определяя наилучшую из них: это — такая последовательность, у которой последний элемент — наименьший возможный. В этой строке наилучшая подпоследовательность длины 1 есть элемент 1, наименьший элемент последовательности. Таким образом, мы приходим к следующей идее: предположим, что мы знаем последний элемент наилучшей подпоследовательности длины k в пройденной части для любого значения k от 1 и вплоть до максимального значения m.

Новый рассматриваемый элемент изучается с точки зрения возможности его присоединения к концу подпоследовательности длины k, чтобы превратить ее в подпоследовательность длины $k+1$. Покажите, что если это возможно, то эта новая последовательность лучше, чем предыдущая подпоследовательность длины $k+1$. Может случиться также, что этот новый член оказывается меньше элемента, образующего подпоследовательность длины 1. Тогда он дает лучшую, чем предыдущая, подпоследовательность длины 1.

Таким образом, вы получаете алгоритм, в котором для любого элемента рассматриваемого вектора нужно искать в таблице последние элементы наилучших подпоследовательностей, и размер этой таблицы равен m. Покажите, что эта таблица упорядочена. Осуществите в ней поиск места рассматриваемого элемента вектора с помощью дихотомического поиска *) и вы получите алгоритм порядка $n \ln n$.

Головоломка 36.

Вы можете вдохновиться решением предыдущей задачи. Нужно пробежать одну из двух цепочек символ

---

*) См. головоломку 29. — *Примеч. пер.*

за символом. Предположим, что мы ее пробежали до некоторого i включительно. Нужно осуществить регистрацию лучших из наиболее длинных слов в порядке возрастания длин, содержащихся в пройденном куске рассматриваемой цепочки и во второй цепочке в целом. Как определить наилучшее слово длины k? Скажем, что это — такое слово, которое имеет наибольшие шансы оказаться продолжаемым, следовательно, такое слово, у которого положение последнего символа во второй цепочке минимально. Это приводит к рассмотрению того, насколько важно знать положение символов во второй цепочке и, следовательно, к заданию наилучших слов списком из положений в цепочке (например, с помощью конкатенации совпадающих с ними символов во второй цепочке).

Бесспорной выглядит трудность, связанная с тем, что одна и та же буква может встречаться во второй цепочке несколько раз. Их нужно рассмотреть все, но их нельзя смешивать между собой. Я уверен, что это вас надолго не задержит.

Больше я вам ничего не сообщаю. Ищите дальше сами...

Головоломка 37.

Вы можете рассмотреть задачу самым простым способом. Пусть задан прямоугольник — координатами x_1, y_1 и x_2, y_2 верхней левой и нижней правой вершины соответственно. Мы выясняем, является ли этот прямоугольник белым (нет ли внутри черной клетки), и если да, то измеряем его площадь.

Мы проделываем это для x_1, y_1, пробегающих все игровое поле, а x_2, y_2 должны удовлетворять неравенствам $x_2 \geqslant x_1$, $y_2 \geqslant y_1$ и пробегать часть игрового поля, удовлетворяющую этим неравенствам.

Так как для каждого прямоугольника вы должны пробежать его по всей его площади целиком, то порядок роста программы есть n^4. Но вы можете улучшить программу уже здесь, не рассматривая такие точки x_1, y_1, которые не могут дать площади прямоугольника, превосходящей уже найденный максимум (это — близкие к правому краю или к нижнему краю точки игрового поля).

Вы можете сделать еще лучше, задав лучшую информацию. Предположим, например, что у вас есть вектор размерности n, — скажем вектор l такой, что $l[i]$ есть число последовательных белых полей на строке i, начиная со столбца j. Тогда вы можете легко найти площади белых прямоугольников, одна из вершин которых находится в точке $x_1 = j$, $y_1 = i$. Нисколько не более трудно перей-

ти и от вектора l для столбца j к вектору, связанному со столбцом $j+1$.

Этих указаний должно быть достаточно для того, чтобы вы сумели получить хороший алгоритм.

Головоломка 38.

Очевидно, что мы очень многого не знаем. Следовательно, нужно тщательно прочесть условие и выделить все данные. Невозможно, чтобы на каждый вопрос решительно все ученики ответили неправильно, потому что если бы это случилось, то они все получили бы 0. Следовательно, на каждый из вопросов есть правильный ответ, который либо является одним из чисел, входящих в ответы учеников, либо другим числом (и тогда более или менее все равно каким).

Таким образом, правильный ответ на первый вопрос может быть одним из чисел

8 12 16 20 и другим числом, скажем 24,

чтобы ответы образовывали арифметическую прогрессию с разностью 4. Сделаем то же самое для других вопросов. Таким образом, вы получите, например:

$R1$:	8	12	16	20	24
$R2$:	12	14	16	18	
$R3$:	10	12	14		
$R4$:	16	18	20	22	24

Исследуем все полученные из оценок четверки чисел, отводя по строчке для каждой из них. Они образуют 5*4*3*5 = 300 строк. Для каждой из них ваша программа смотрит, сколько учеников получило 0, и запоминает только те четверки чисел, для которых один и только один ученик получил 0 (это дано в условии). Заметьте к тому же, что вам сообщено, что ответом на один из вопросов должна быть площадь поверхности куба с целым ребром, следовательно, число вида $6n^2$, возможные значения которого 6, 24... Ни один из ответов не имеет вида $6n^2$ с целым n. Следовательно, мы должны получить, что в выделенных четверках есть одна или несколько четверок, у которых хотя бы одна компонента имеет значение, не предложенное ни одним из учеников. Ваша программа легко их найдет (такой набор в точности один). На этом основании мы узнаем правильность всех ответов на вопросы, остальное просто.

При всем том, это — головоломка для начинающих...

Головоломка 39.

Эта головоломка сопротивлялась мне много дней и была для меня очень поучительной. В условии сказано, что эта программа должна выполняться за время вычисления, пропорциональное n. Следовательно, и речи нет о том, чтобы исследовать все суммы подпоследовательностей вектора, чтобы выбрать из них наилучшую. Нужно исхитриться. Так же, как мы здесь уже упоминали, ответ может состоять в получении свойств подпоследовательности с максимальной суммой.

Я совершил ошибку, пойдя по этому пути. Я сказал себе: назовем $S(i, j)$ сумму элементов вектора с номерами от i до j:

$$S(i, j) = a_i + a_{i+1} + \ldots + a_{j-1} + a_j.$$

Если для некоторой пары i, j эта сумма максимальна, то отсюда следует

$$S(i, j) > S(i+1, j)$$

и, следовательно, $a_i > 0$. Точно так же $a_i + a_{i+1} > 0$.

Если обобщить любое «начало» (левая часть) $S(i, j)$ положительно, и точно так же любой «конец» положителен. Можно продолжать:

a_{i-1} отрицателен...

И я таким образом не получил ничего. Это не означает утверждения, что на этом пути нельзя найти решения. Это я его не нашел.

Как я уже говорил, вы можете обратиться к математике за помощью в решении вашей задачи по информатике *). Но у информатики есть и свой собственный творческий дух. Почему бы ему не довериться? Эта задача

---

*) В математике для решения этой задачи есть полезная формула Ньютона — Лейбница: $\int_p^q f(x)\,dx = F(q) - F(p)$, где F — функция, определяемая условием $F'(x) = f(x)$, $x \in [p, q]$. Впрочем, все эти интегралы нам не понадобятся, так как у этой формулы есть гораздо более простой аналог для сумм: $a_i + a_{i+1} + \ldots + a_j = b_j - b_{i-1}$, где последовательность $\{b\}$ определяется условием, что $b_k - b_{k-1} = a_k$ для любого k от 1 до n. Последовательность $\{b\}$ легко построить: $b_0 = 0$, $b_{k+1} = b_k + a_{k+1}$. Для последовательности $\{b\}$ задача ставится так: найти такие $i < j$, что разность $b_j - b_i$ максимальна. С этой задачей уже легче справиться.— *Примеч. ред.*

сбивает вас с толку по причине ограничений на сложность алгоритма. Забудем их. Если вам сказано, что нужно решить задачу, и вам предоставлена свобода вплоть до максимальной сложности, что вы будете делать? Вы составите таблицу $S(i, j)$ для $i = 1, \ldots, n$ и $j = i, \ldots, n$. В этой таблице вы возьмете максимальный элемент.

Чтобы помочь вам, я предлагаю вам рассмотреть следующий вектор:

3 4 —8 2 —3 7 5 —6 1

Образуйте треугольную таблицу чисел $S(i, j)$ и запишите ее. Посмотрите, как каждая строчка образуется из предыдущей. Вы увидите, что только три строчки могут содержать максимальное S и, кроме того, не во всей их полной длине. В этом примере максимум нужно искать среди

(1, 1 : 3), (4, 4 : 5), (6, 6 : 9).

Следовательно, есть в точности n значений S, которые нужно рассматривать. Таким способом вы и получаете алгоритм, линейный по n.

Закончить предоставляю вам.

ЧАСТЬ III

И ЕСЛИ ВЫ ВСЕ ЕЩЕ НЕ НАШЛИ РЕШЕНИЯ

Многие игры или головоломки уже не требуют никаких дополнительных пояснений. Но некоторые из них еще могут вам сопротивляться. Поэтому следует сказать вам все...

1. СЛУЧАЙНЫЕ ЧИСЛА

Головоломка 1.

Первая стратегия. Нужно сравнить u_{2i} и u_i. Они равны, если $2i = i + kp$ для целого k, следовательно, если i делится на p. Кроме того, i должно превосходить r. Следовательно, нужно искать наименьшее кратное p, большее или равное r.

Положим $v_i = u_{2i}$. Тогда

$$v_{i+1} = u_{2i+2} = f(f(u_{2i})) = f(f(v_i)).$$

Если вы начинаете u с $u_1 = a$, то вы начинаете v с $v_1 = f(a)$.

Таким образом, получаем начало программы:

```
u := a; v := f (a)
ПОКА u ≠ v ВЫПОЛНЯТЬ
   u := f (u); v := f (f (v))
ВЕРНУТЬСЯ
```

Теперь вы получили два равных элемента. Чтобы получить период, нужно пройти интервал между полученными числами — например, начиная с u — считая число элементов:

```
p := 1; w := f (u)
ПОКА w ≠ u ВЫПОЛНЯТЬ
   w := f (w); p := p + 1
ВЕРНУТЬСЯ
```

Мне пришлось рассказать вам все...

Вторая стратегия. Начните с $d = 1$ и $h = 1$. Если вы не находите периодичности в интервале от $d + 1$ до

$d + h$ (сравнивая u на этом интервале со значением u на элементе d, сохраняемым в некоторой переменной, например, x), возьмите значение u в $d + h$ в качестве нового значения x, $d + h$ в качестве нового d, и удвойте h.

Вы непосредственно получаете период. Тщательно подсчитайте количество вычислений f в каждом из этих двух алгоритмов. Второй способ определенно лучше.

Игра 4.

Если вы представляете игровое поле прямоугольной таблицей, то перемещение обозначается изменением координат точки: добавлением или вычитанием чисел 1 или 2. Я разместил эти добавляемые количества (целые числа со знаком) в два вектора DX, DY из 8 элементов. Одно направление перемещения задается номером поля в этой таблице, следовательно, целым числом от 1 до 8.

2. ИГРЫ С ЧИСЛАМИ

Головоломка 3.

Остановитесь, когда вы получите 5 в качестве цифры единиц с нулем «в уме».

Головоломка 4.

Представленный здесь алгоритм эквивалентен алгоритму, который можно найти в старых книгах по арифметике, и который действует на целые числа, разбитые на куски по 2 цифры в каждом куске. Вы можете либо разыскать доказательство в этих книгах, либо посмотреть в моей книге «Основы программирования», как можно доказать, что программа, реализующая этот алгоритм, действительно вычисляет квадратный корень. Но это рассуждение слишком сложно, чтобы воспроизводить его здесь.

Лично я работаю по основанию 10. Я представляю числа цепочками цифр. Присоединить 1 справа легко: это просто конкатенация. Сдвинуть вправо легко: используется индекс, сообщающий, начиная с какой позиции нужно урезать. Именно этот индекс и изменяется. Складывать с 2 легко, так как может быть не более одного переноса. Единственная тонкая операция — вычитание. Не проводите сравнения перед вычитанием: оно стоит так же дорого, как и само вычитание. Сделайте копию той части, которая должна была бы быть изменена при вычитании, и если вы обнаружите, что вы не можете осуществить вычитание,— возьмите сохраненное значение.

Головоломка 5.

Задайте три индекса и три значения: i_2, i_3, i_5, x_2, x_3, x_5. Число i_2 есть индекс элемента последовательности, который, будучи умноженным на 2, дает подходящего кандидата на роль ближайшего значения (иначе говоря, удвоение числа с индексом $i_2 - 1$ дает число, которое содержится в уже сформированной части последовательности, но удвоение числа с индексом i_2 дает число, которое в сформированной части не содержится). Число x_2 получается удвоением числа с индексом i_2. Вы определяете аналогично i_3 и x_3, заменяя «удвоение» на «утроение» (произведение на 3 числа с индексом $i_3 - 1$ содержится в построенной части последовательности, а число x_3 — утроенное число с индексом i_3 — в ней не содержится). Наконец, вы делаете то же самое для i_5 и x_5. Ближайшее число в последовательности есть наименьшее из чисел x_2, x_3, x_5. Назовем его x. Если $x = x_2$, то i_2 увеличивается на 1 и x_2 пересчитывается. То же самое для i_3 и i_5.

Головоломка 7.

Возьмем $n = 3n' + 2$. Тогда $(2n - 1)/3 = 2n' + 1$.

По общему правилу, непосредственно следующий за нечетным числом $2n' + 1$ элемент равен $(3(2n' + 1) + 1)/2 = 3n' + 2$.

Если n дает n' при переходе (p, q), $q > 1$, т. е. если n имеет вид $n = (2^p (2^q n' + 1)/3^p) - 1$, то

$$n'' = (n - 1)/2 = (2^{p-1} (2^q n' + 1)/3^p) - 1.$$

Как и следовало ожидать, это имеет в точности тот смысл, что если деление на 3^p можно выполнить нацело, то в связи с этим возникает соотношение между (p, q) и n'.

Если n'' увеличить на 1, а затем умножить на $3^{p-1}/2^{p-1}$, то получится $(2^q n' + 1)/3$.

Тогда нужно уменьшить результат на 1: получим $(2^q n' - 2)/3$. Но это число делится на 2, так что с помощью перехода $(p - 1, 1)$ число n'' дает

$$(2^{q-1} n' - 1)/3.$$

По общим правилам получаем

$$3((2^{q-1} n' - 1)/3) + 1 = 2^{q-1} n',$$

а затем n', что и доказывает наше утверждение.

Если вы примените это правило перехода к $4k + 1$, то нужно добавить 1, что дает $4k + 2$, делящееся на 2,

но не на 4. Делим на 2 и умножаем на 3, что дает $6k+3$. Уменьшаем на 1 и затем делим на 2, и получается $3k+1$.

Если k нечетно, то это — элемент, следующий за k; так что за числом вида $4k+1$ с k нечетным следуют те же величины, что и за k.

Если k четно, то $4k+1$ дает $3k+1$.

Если существует цикл с единственным переходом p, q, т. е.

$$n = (2^p (2^q n + 1)/3^p) - 1,$$

то это возможно только в случае, когда существует такая пара p, q, что число

$$(3^p - 2^p)/(2^{p+q} - 3^p)$$

— целое. Мы показали, что такой пары (p, q) нет.

Головоломка 10.

9*АВСДЕ + АВСДЕ = 10*АВСДЕ, что можно записать как АВСДЕ0. Отсюда получаем зашифрованное сложение:

```
+ FGHIJ
  ABCDE
 ------
 ABCDE0
```

Это показывает, что $A = 1$. Далее, $J + E$ не может быть нулем, следовательно, $J + E = 10$ и для I есть коечто «в уме». Сумма $F + A$ дает AB с $A = 1$, так что сумма $F + 1$, к которой, может быть, добавлено что-то «в уме», должна дать число, большее 9. Это может быть только в случаях $1 + 8 + 1 = 10$, $9 + 1 = 10$ или $1 + 9 + 1 = 11$. Но, так как $B \neq A$, то $B = 0$.

Тогда в сумме $G + B$ рассмотрим цифру C как цифру единиц. Так как $B = 0$, то это означает, что для G «в уме» кое-что есть (потому что $G \neq C$).

Отсюда получаем схему операции сложения:

$$\begin{array}{cccccc} & k' & 1 & k & 1 & \\ & F & G & H & I & J \\ & 1 & 0 & C & D & E \\ 1 & 0 & C & D & E & 0 \end{array}$$

Запишем, что $A + B + C + D + E + F + G + H + I + J = 45$,

$A = 1$, $B = 0$.

Запишем пять операций сложения с учетом переносов в старший разряд:

$$J + E = 10,$$
$$1 + I + D = 10k + E,$$
$$k + H + C = 10 + D,$$
$$1 + G + B = 10k' + C,$$
$$k' + F + A = 10.$$

Сложим их все. Вам остается

$$C + D + E = 17 - 9\,(k + k').$$

Но $C + D + E$ не может быть меньше, чем $2 + 3 + 4 = 9$, и не может быть больше, чем $6 + 7 + 9$ (если $F = 8$ и $k' = 1$). Не может быть, чтобы у вас одновременно выполнялись соотношения $k = k' = 1$ (что давало бы отрицательную сумму $C + D + E$). Но не может быть и равенства $k + k' = 1$, так как тогда было бы $C + D + E = 17 - 9 = 8$, что слишком мало. Следовательно, $k = k' = 0$. Составим окончательную систему

$$J + E = 10,$$
$$I + D + 1 = E,$$
$$H + C = 10 + D,$$
$$G + 1 = C,$$
$$F = 9.$$

Закончите вы с помощью программы.

Головоломка 11.

Обозначим через a_i цифры исходного числа, b_i — цифры результата, k_i — цифры «в уме»:

$$3a_i + k_i = b_i + 10k_{i+1}.$$

Сумма всех a_i равна 45, как и сумма всех b_i. Обозначим через K сумму всех k_i:

$3*45 + K = 45 + 10*K$ дает $K = 10$.

Мы знаем, что дает «в уме» каждая цифра:

1 дает 0, 2 дает 0, 3 дает 0 или 1 в зависимости от того, что хранится «в уме» над 3.

4 дает 1, 5 дает 1, 6 дает 1, потому что не может случиться $3*6 + 2$, что давало бы «в уме» 2, но цифру единиц 0;

7, 8 и 9 дают 2.

Для того, чтобы сумма величин «в уме» была равна 10, нужно, чтобы 3 давало 1 «в уме». Так как $3*3 + 1$ (с цифрой единиц, равной 0) случиться не может, то нужно, чтобы «в уме» над 3 было 2. Следовательно, 3 стоит

слева от 7, 8 или 9. В частности, 3 не может стоять на правом конце.

Остальное просто, если вы будете следовать методу, указанному в разделе «Условия». Вот таблица:

В уме	0	1	2
1	7	0	3
2	4	7	0
3	1	4	7
4	8	1	4
5	5	8	1
6	2	5	8
7	9	2	5
8	6	9	2
9	3	6	9

Потребуем, чтобы 9 было справа; следовательно, вычеркнем 9 из этой таблицы, оставив его только в столбце, соответствующей тому, что «в уме» 0. Цифра 3 требует 2 «в уме», чтобы дать 1. Вычеркнем остальные 3 в таблице. Цифра 9 не может быть получена иначе как с помощью 6 и 1 «в уме». Другие 6 вычеркиваем. Цифра 8 получается из 2 при 2 «в уме». Нужно взять 3 числа в первом столбце, так что нужно еще одно не равное ни 2, ни 3. Их нужно 4 в среднем столбце, так что нужно еще 3 числа, не равных 6, которые нужно взять среди цифр 7, 4, 1, 8, 5. Два последних числа должны быть взяты из столбца с нулем «в уме». Когда эти числа среди всех возможных будут выбраны, останется расположить их в соответствии с тем, что должно быть для них «в уме». Эту программу сделать легко.

Головоломка 12.

Если число $a_1a_2 \ldots a_p$ (представленное как последовательность цифр) кратно 3, то и $a_1 + a_2 + \ldots + a_p$ кратно 3. Сумма кубов цифр равна

$$a_1^3 + a_2^3 + \ldots + a_p^3.$$

Нужно показать, что это число также кратно 3. Действуйте по индукции по числу слагаемых. Предположим, что для $p = n - 1$ членов

$$a_1^3 + a_2^3 + \ldots + a_p^3 = (a_1 + \ldots + a_p)^3 \text{ по модулю } 3;$$

тогда равенство

$$(a_1 + \ldots + a_p + a_n)^3 = (a_1 + \ldots + a_p)^3 + a_n^3 + 3\,(\ldots)$$

доказывает наше утверждение для n слагаемых.

Возьмите число с k цифрами. Сумма кубов его цифр ограничена величиной $k*9^3$. Но исходное число не может быть меньше, чем 10^{k-1}. Следовательно, достаточно, чтобы 10^{k-1} было больше, чем $k*729$, что очевидным образом выполняется при $k = 5$. Но эта оценка слишком пессимистична.

Головоломка 14.

Число, полученное при обращении порядка цифр, равно

$$1000d + 100c + 10b + a,$$

и разность этих двух чисел равна

$$999(a - d) + 90(b - c).$$

Числа a, b, c, d были расположены в невозрастающем порядке, и они не все равны между собой, так что a строго больше d и $a - d$ не равно нулю. Все остальное просто.

Головоломка 16.

Единственное, что до сих пор еще не сказано — это способ определять, становится ли последовательность периодической. Метод Полларда был основан на первой стратегии. Мы выясняем, существует ли a_i с $a_{2i} = a_i$. Но вычисление $f(x) = x^2 - 1$ по модулю n — дорогое вычисление. Брент улучшил этот метод, предложив использовать вторую стратегию.

Головоломка 17.

Эта программа основана на следующих результатах:

если b нечетно, n четно, то n делится на b тогда и только тогда, когда $n/2$ делится на b;

нечетное n делится на b тогда и только тогда, когда $n - b$ делится на b. Но $n - b$ четно.

Для $n = 2^{77} - 3$ и $b = 7$ вы получаете:

Число n нечетно. Рассматриваем $n - b = 2^{77} - 10$. Оно делится на 2: получаем $2^{76} - 5$.

Это число нечетно: $(2^{76} - 5) - 7 = 2^{76} - 12$.

Делим на 4: $2^{74} - 3$.

Получаем ту же самую задачу, в которой показатель уменьшен на 3. Так как $77 = 3*25 + 2$, то мы таким образом доходим до $2^2 - 3 = 1$, которое не делится на 3. Вряд ли вас слишком утомит доказательство того, что $2^n - 3$ никогда не делится на 7...

Головоломка 18.

Я не в состоянии рассказать вам, как я получил эту программу, это — очень долгая история, связанная с разложением целых чисел на множители. Может быть, ког-

да-нибудь я ее и опубликую. Следовательно, будем разбираться в том, что нам дано — в тексте программы.

Начнем с нечетного n. В соответствии с инициализацией программы $n = 4p - 1$, где p четно. В противном случае уже последует ответ «НЕТ». Следовательно, рассмотрите нечетное n, являющееся полным квадратом и, следовательно, квадратом нечетного числа $2k + 1$:

$$(2k + 1)^2 = 4k^2 + 4k + 1 = 4k(k + 1) + 1.$$

Так как $k(k + 1)$ — произведение двух последовательных целых чисел, и из двух последовательных целых чисел всегда есть хотя бы одно четное число, получаем простой, но интересный результат: любой квадрат нечетного числа сравним с 1 по модулю 8. Таким образом, при n отличном от 1 по модулю 8 инициализирующая часть программы выводит, что n не является точным квадратом.

Посмотрим теперь, что происходит внутри цикла. Делим p на 2, и если результат четен, мы удовлетворяемся тем, что умножаем a на 2. При этом действии произведение $a*p$ остается постоянным. Поэтому кажется вероятным, что в цикле существует инвариантная величина, запись которой содержит $a*p$ в предположении, что p четно.

Если после деления p на 2 результат оказывается нечетным, то мы вычитаем из этого результата $a/2 + b$. Обозначим новые значения a, b, p через a', b', p' соответственно:

$$a' = 2*a,\ p' = p/2 - a/2 - b,\ b' = a + b.$$

Для этих значений получаем:

$$a'*p' = a*p - a^2 - 2a*b = a*p - (a + b)^2 + b^2 = a*p - b'^2 + b^2.$$

Это, наконец, дает

$$a'*p' + b'^2 = a*p + b^2.$$

Инвариантной величиной цикла оказывается, таким образом, сумма $ap + b^2$, причем p остается четным. Это обеспечивается тем, что в случаях, когда $p/2$ нечетно, мы вычитаем нечетные b из нечетного $p/2$. Что касается b, то он нечетен потому, что он начинается со значения 1 и к нему прибавляются только четные значения a.

В начале $a = 4$, p (целая часть дроби $(n - 1)/4$) четно, $b = 1$, так что $ap + b^2 = n$.

Наконец, a, начиная с 4, умножается на 2 при каждом прохождении цикла; b начинается с 1, которое меньше соответствующего начального $a = 4$.

Тогда при переходе от a, b, p к a', b', p' либо

$b' = b$, $a' = 2*a$, так что если $b < a$, то и $b' < a'$;

либо

$b' = a + b$, $a' = 2*a$, что также сохраняет справедливость отношения $a' < b'$.

Следовательно, вот ситуация, которую цикл оставляет инвариантной:

$n = a*p + b^2$;
a — степень двойки,
p четно,
b нечетно, $b < a$.

Кроме того, мы знаем, что при выходе из цикла $p < a$.

Если p равно нулю, то $n = b^2$. Тогда мы видим, что n — квадрат числа b, которое выводится, и все закончено.

Но n может оказаться полным квадратом и тогда, когда p не нуль. Попробуем рассмотреть все возможные случаи. Положим $n = r^2$ (r нечетно).

Соотношение

$r^2 = ap + b$

дает

$r^2 - b^2 = ap$.

Положим $r + b = 2u$, $r - b = 2v$ (r и b нечетны).

Отсюда получаем $4uv = ap$.

Поскольку $r = u + v$, где r нечетно, получаем, что u и v не могут быть числами одинаковой четности, так что одно из них четно, а другое нечетно. Так как a является степенью двойки, то нечетный сомножитель относится к p. Выявим его, полагая $p = s2^t$, где s нечетно, а $t \geqslant 1$ (p четно).

Напомним, что $a = 2^k$. В этих обозначениях

$4uv = ap = s2^{k+t}$, $uv = s2^{k+t-2}$.

Возможные решения для пары u, v имеют вид пар

$s'2^{k+t-2}$, s'',

где $s's'' = s$.

Покажем сначала, что s'' — меньший из этих двух элементов пары. Вследствие $t \geqslant 1$ имеем $k - t \leqslant k + t - 2$.

Вследствие $p < a$ последовательно выводим

$$s2^t < 2^k,$$
$$s's''2^t < 2^k,$$
$$s's'' < 2^{k-t} \leqslant 2^{k+t-2} \leqslant s'^2 2^{k+t-2}$$

(потому что s' нечетен и не меньше 1).

Следовательно, нужно взять $u = s'2^{k+t-2}$, $v = s''$.

Покажем теперь, что нужно обязательно взять $s' = 1$, $s'' = s$. По выбору u и v

$$b = 2^{k+t-2}s' - s'' < a = 2^k.$$

Отсюда получаем:

$$s'' > 2^{k+t-2}s' - 2^k,$$

и, так как $t \geqslant 1$:

$$s'' > 2^{k-1}s' - 2^k,$$
$$s = s's'' > 2^{k-1}s'^2 - 2^k s = 2^{k-1}s'(s' - 2).$$

Вследствие $p = s2^t < a = 2^t$ выводим $s < 2^{k-t} \leqslant 2^{k-1}$.

Объединим два полученных неравенства:

$$2^{k-1}s'(s' - 2) < x < 2^{k-1}, \text{ поэтому } s'(s' - 2) < 1.$$

Единственное нечетное число s', удовлетворяющее этому соотношению, это $s' = 1$. Следовательно, у нас остается единственная возможность:

$$u = 2^{k+t-2}, \; v = s,$$
$$b = u - v = 2^{k+t-2} - s < a = 2^k,$$
$$s > 2^{k+t-2} - 2^k.$$

Так как $s < 2^{k-t}$, то t должно быть таким, чтобы

$$2^{k-t} > 2^{k+t-2} - 2^k.$$

Поскольку t должно быть строго положительно, то его единственными возможными значениями являются $t = 1$ и $t = 2$.

При $t = 1$ имеем

$$p = 2s, \quad b = 2^{k-1} - s = a/2 - p/2.$$

Следовательно, если $2b = a - p$, то n — квадрат числа $(a + p)/2 = a - b$.

При $t = 2$ имеем

$$p = 4s, \; b = 2^k - s = a - p/4.$$

Следовательно, если $p = 4(a - b)$, то n — квадрат числа $a + p/4 = 2a - b$.

Этим исчерпываются случаи, когда n может быть полным квадратом.

Можно спросить себя, могут ли эти различные случаи действительно осуществляться. Заметим, что при вступлении в цикл у нас $b = 1$, $a = 4$. После этого b может быть изменено добавлением a, т. е. кратным числа 4. Следовательно, b остается сравнимым с 1 по модулю 4.

В трех возможных случаях:

$$p = 0, \quad r = b,$$
$$p = a - 2b, \quad r = a - b,$$
$$p = 4(a - b), \quad r = 2a - b,$$

первый случай — единственный, в котором квадратный корень из n сравним с 1 по модулю 4; два других дают квадратный корень, сравнимый с 3 по модулю 4.

При выходе из цикла равенство

$$n = ap + b^2$$

с учетом соотношений $p < a$, $b < a$ дает $n < 2a^2$ и, следовательно, при выходе из цикла $a^2 > n/2$.

Равенство

$$ap = n - b^2$$

дает $p = (n - b^2)/a < n/a$.

Если окажется, что $n/a < a$, то непременно $p < a$ и цикл закончен. Чтобы цикл остановился, необходимо, чтобы $a^2 > n/2$, и цикл заведомо останавливается, если $a^2 > n$.

Следовательно, все зависит от положения n по отношению к степеням двойки. Существует такое целое n, что

$$4^q < n < 4^{q+1}.$$

Возможны два случая. Во-первых, может выполняться неравенство

$$4^q = 2^{2q} < n < 2^{2q+1},$$

и тогда для $k = q$ число $a^2 = 2^{2q} > n/2$ может быть значением остановки, но в этом нет уверенности. С другой стороны, если

$$2^{2q+1} < n < 2^{2q+2},$$

то единственное значение a, удовлеторяющее условию $a^2 > n/2$, есть $a = 2^{q+1}$, и для этого значения имеем

$a^2 > n$, что гарантирует остановку. Поскольку $r = a - b$, то $a = r + b > r$ и, следовательно, $a^2 > n$.

Во втором случае

$$r = 2a - b \text{ и } b < a, \text{ откуда } a < 2a - b = r.$$

Таким образом, все три распознаваемые программой случая являются единственными возможными исходами программы, и каждый из них может произойти.

Таким образом, перед нами — очень забавный алгоритм, который дает значение квадратного корня, и который определяет случай, когда n не является корнем, но в этом случае не дает никакой дополнительной информации.

Программа заведомо останавливается при $a = 2^{q+1}$, так что число шагов цикла не больше $q - 1$ (начиная с 4), причем q — логарифм квадратного корня из n по основанию 2. Таким образом, получилась программа порядка $\ln n$, что дает ту же сложность, что и обычный алгоритм, действующий кусками по две цифры. Но для этого последнего алгоритма нужен еще первый цикл, чтобы найти порядок величины n.

Головоломка 19.

Соотношение $f(a, b) = f(b, a)$ следует из самой инициализации p и q:

$$p := \max(a, b); \quad q := \min(a, b).$$

Эти две функции симметричны по a и b, и поэтому точно так же симметрична f. При анализе программы мы ограничены действиями, происходящими внутри цикла. Величины r и s являются вспомогательными переменными, которые не оставляют никакой проблемы. Трудность вызывают преобразования, проделываемые над p и q. Чтобы ясно увидеть эту трудность, осуществим введение новых переменных без разрушения старых. Перепишем наш цикл:

```
ПОКА q ≥ eps ВЫПОЛНЯТЬ
r := (q/p)²;   s := r/(r + 4)
p' := (2 * s + 1) * p;   q' := s * q
p := p';   q := q'
ВЕРНУТЬСЯ
```

Рассмотрим действия этой программы, производимые над данными a, b с одной стороны и над ac, bc с другой.

Когда мы входим в цикл, то и p, и q умножаются на c при переходе от первого вычисления ко второму.

Это не меняет величины r и, следовательно, не меняет величины s. Таким образом, p и q в этих вычислениях умножаются на одни и те же сомножители, так что значения p', q' во втором вычислении получаются из значений p, q в первом вычислении умножением их обоих на c. Следовательно, мы еще раз входим в цикл при том же соотношении между входными данными; следовательно, это соотношение будет иметь место при каждом входе в цикл, и, следовательно, также и на выходе из цикла. Отсюда получаем, что $f(ac, bc) = cf(a, b)$.

Выполнение программы для вычисления $g(x) = f(x, 1)$ с $x = 1$ и $eps = 10^{-5}$ дает мне результат, равный 1.4142.

Дальше считать бесполезно, это $\sqrt{2}$.

Я немедленно изменяю программу, чтобы она выполняла вывод не только величины g, но также и g^2. Я получаю:

x	$g^2(x)$
1	2
2	5
3	10
4	17

Нет возможности сомневаться: $g(x) = \sqrt{x^2 + 1}$.

Перенося эту формулу в соотношение между f и g, мы видим, проделав все вычисления, что

$$f(a, b) = \sqrt{a^2 + b^2}.$$

«Осталось только» доказать это. Мы не можем доверять заверениям программистов, утверждающих, что их программа делает то-то и то-то. При входе в цикл p и q имеют значения a и b в каком-то порядке, поэтому

$$p^2 + q^2 = a^2 + b^2.$$

Что происходит с величиной $p^2 + q^2$ после изменений, которым p и q подвергаются в цикле? Вычислим $p'^2 + q'^2$:

$$p'^2 + q'^2 = (2s + 1)^2 p^2 + s^2 q^2 = s^2(4p^2 + q^2) + 4sp + p^2.$$

Вычислим s:

$$r = q^2/p^2, \quad s = r/(r + 4) = q^2/(q^2 + 4p^2),$$

откуда, наконец,

$$s(4p^2 + q^2) = q^2.$$

Возвращаясь отсюда к предыдущему соотношению, получаем

$$p'^2 + q'^2 = sq^2 + 4sp^2 + p^2 = \\ = s(4p^2 + q^2) + p^2 = p^2 + q^2.$$

Таким образом, все кончено... Это соотношение гарантирует, что $p^2 + q^2$ является инвариантом цикла. При каждом входе в цикл выполняется соотношение

$$p^2 + q^2 = a^2 + b^2.$$

При выходе из цикла

$$p^2 + q^2 = a^2 + b^2, \text{ причем } q < eps.$$

Отсюда следует, что

$$p^2 = (a^2 + b^2) * (1 - q^2/(a^2 + b^2)).$$

Сразу получаем, что

$$p = \sqrt{a^2 + b^2}$$

с относительной ошибкой $eps^2/(2 * (a^2 + b^2))$.

Чтобы получить точность до 10^{-5}, совершенно ненужно брать $eps = 10^{-5}$; более чем достаточно $eps = 0.004$.

Эта программа сходится очень быстро.

8. ИГРЫ БЕЗ СТРАТЕГИИ

Игра 13.

Задача о наиболее длинном взятии не имеет однозначного решения. Вот как ее сделал я — с учетом моих привычек в программировании и упрощений, предоставляемых моим микрокомпьютером.

Я представил всю игру одной-единственной цепочкой символов с кодами возврата каретки, расположенными надлежащим образом — так, чтобы визуализация игры сводилась к визуализации этой цепочки без какого-либо дополнительного исследования. Куры обозначаются в этой цепочке присвоенными им буквами, лисы — буквами X, пустые игровые поля обозначаются точками. Остальные символы (пробелы или возврат каретки) не отвечают никаким используемым игровым полям. Я добавил в начале и в конце по строчке пробелов, чтобы не было необходимости изучать возможность некоторых перемещений на границе игрового поля.

Я не храню положений лис с помощью двух переменных. Я отыскиваю их положение в цепочке, представляю-

щей игру, с помощью функции «положение» языка LSE. Это — существенная деталь. Поиск наиболее длинного взятия я осуществляю итеративно. Я образую две цепочки:

— одна из них содержит список кур, уже взятых при исследовании данного пути (это — последовательность букв взятых кур),

— вторая цепочка содержит дуплеты: положение в игре и рассматриваемое направление (мы осуществляем взятие, исходя из положения x и двигаясь в направлении, обозначенном через i).

Находясь в положении x и в направлении i я смотрю. есть ли кура на поле $x + d[i]$. Если ее нет, то в этом направлении никакое взятие невозможно. Если же такая кура есть, то я смотрю, не содержится ли буква этой куры в цепочке уже взятых кур. Если содержится, то в этом направлении ничего не сделаешь. Если же эта кура еще не взята, то я проверяю, действительно ли поле $x + 2 * * d[i]$ содержит именно точку — в противном случае никакого взятия нет. Действуя таким образом, я не сталкиваюсь ни с какими трудностями на краях (там есть предохранительная строка, и она не содержит ни одной куры).

Если взятие оказывается возможным, я присоединяю его характеристики к обеим цепочкам, продвигаюсь на новую позицию и возобновляю изучение взятий, исходя из этого нового положения. Я не изменяю состояния игры, за исключением того, которое относится к начальному полю отправления лиса (на этом поле лис может оказаться снова. Напротив, из соображений четности мы не можем прийти на поле, занимаемое какой-либо из взятых кур).

Когда оказывается, что мы достигли поля, исходя из которого уже никакое дальнейшее взятие невозможно, я сравниваю длину цепочки взятых кур с наиболее длинной уже сохраняемой цепочкой и выбираю лучшую из них (нужно смотреть и на цепочку дублетов, чтобы осуществить взятие, обновляя состояние игры, как только наиболее длинное взятие будет определено). Затем я отменяю последнее взятие (совершенное в этих двух цепочках) и перехожу к следующему направлению, исходя из последнего положения. Никакой проблемы с временем вычислений на моем микрокомпьютере не возникает, даже наоборот. Часто нужно добавлять замедляющий цикл, чтобы предоставить игроку время увидеть, что происходит...

Игра 19.

И здесь тоже решение неединственно. Вот одно из них, хорошо приспособленное к используемой мною машине, на которой деления на 2 не бесплатны (на мой взгляд, выполняются слишком долго). Нам известна верхняя граница числа спичек в каждой кучке, определяемая принятым вами правилом (я взял 4 кучки с по крайней мере 16 спичками). Я рассматриваю двоичные записи числа спичек в кучке, начиная слева. Я задаюсь числом $p = 8$. Если число больше или равно 8, то оно содержит 1 в крайнем левом из возможных положений. Тогда я вычитаю 8 из этого числа и перехожу к $p = 4$.

Сначала я определяю крайнюю левую цифру для числа спичек во всех четырех кучках. Из них я удерживаю только две вещи: четность этого числа (переменная q, вначале равная 0, изменяется на $1 - q$ при каждой встрече с единицей; результат нечетен, если в конце получается $q = 1$); номер последней встреченной кучки, у которой в данном положении встретилась единица. Я исследую таким образом все положения слева направо, пока не встречаю положение, для которого сумма единиц, стоящих в этом положении, нечетна. Тогда я знаю кучку (ту, номер которой был удержан в памяти), у которой в этом положении стоит именно единица. Я убираю из этой кучки желаемое число спичек, чтобы эта единица исчезла (8, если сейчас изучается крайнее левое положение).

Тогда я аналогичным образом исследую оставшиеся положения, за исключением того, что я больше не трогаю номера кучек, из которых я уже брал спички. Для каждой оставшейся позиции, вплоть до крайней правой, я отыскиваю единицы и, если число единиц нечетно, то я изменяю число спичек в выбранной кучке. Чтобы узнать, нужно ли добавить или уменьшить, я сохраняю их число перед изменением в цикле. Если оно больше p, я вычитаю из него p, а если меньше, то я p добавляю (я ставлю 0 вместо 1 и 1 вместо 0). Все это — очень быстрое вычисление, но оно требует немного больше строк в программе. Так вы легко достигнете цели.

Игра 20.

Я ничего не добавляю, потому что эта игра является вариантом нимской игры. На каждой строке есть пустые поля, играющие ту же роль, что и спички в кучках нимской игры. Единственная разница состоит в том, что игрок может отступать. Если вы можете достичь выигры-

вающей позиции (такой, что НИМ-сумма пустых полей между игроками на каждой строчке оказывается равной нулю) и если противник отступает одной из своих шашек, увеличивая число пустот на этой строке, то вы на столько же полей продвигаетесь вперед, восстанавливая таким образом предшествующую — и потому выигрывающую — ситуацию. Противник оказывается немного глубже вбитым в свой угол и приблизившимся к своей гибели. Если в какой-то момент все промежуточные пустые поля пропадут, то шашки оказываются рядом друг с другом, и ваш противник может только отступать. А вы за ним следуете...

Игра 22.

Вот шкала весов, предложенная А.—П. де Лоашем в книге Шварца [SCH]:

0: отрезок замыкает проигрывающий треугольник,

1: отрезок замыкает треугольник, две стороны которого принадлежат моему противнику,

2: отрезок замыкает смешанный треугольник (одна из сторон которого — моя, а другая — моего противника),

3: отрезок соединяет две смешанных вершины (из каждой из них выходит и моя сторона, и сторона моего противника),

4: отрезок соединяет смешанную вершину и вершину, из которой выходит только одна сторона,

5: отрезок соединяет две вершины, каждая из которых принадлежит только одному отрезку, который провел именно я,

6: отрезок соединяет две вершины, каждая из которых принадлежит только одному отрезку, один из которых провел я, а другой — мой противник,

7: отрезок соединяет две вершины, которые не принадлежат проведенным мною отрезкам,

8: отрезок проходит через «чистую» (еще не использованную) вершину,

9: отрезок соединяет чистую вершину с вершиной, не принадлежащей моим отрезкам,

10: отрезок соединяет две чистые вершины.

Можно немного упростить этот список, не увеличивая сколько-нибудь серьезно опасность проиграть.

Игра 23.

Мы приводим список выигрывающих позиций, подразумевая при этом, что если $S(p, q)$ выигрывает, то выигрывает и $S(p, q')$ для всех q', не превосходящих q. Выберем для каждого p наибольшее возможное для него

значение q.

3,1 5,2 8,3 11,1 13,6 16,1 18,2
21,10 24,1 26,2 29,3 32,1
34,16 37,1 39,2 42,3 44,1 47,6 50,1

Игра 24.

Мне кажется, что лучше оценивать комбинацию, вычисляя число черных шашек, а затем число появлений каждого цвета в этой комбинации и, наконец, сумму по всем различным цветам меньших (из значений в обеих комбинациях) чисел, получаемых в сравниваемых комбинациях (это число равно сумме числа белых и черных шашек). Вы сохраняете число появлений каждого цвета в каждой уже испытанной комбинации в таблице с двумя индексами.

Для каждой новой комбинации предложение некоторого цвета в некоторой позиции ведет к следующему:

— для черных шашек вы можете осуществить сравнение с той же позицией в другой комбинации;

— для множества белых + черных шашек: вы получаете еще и другие появления этого цвета, и у вас хранится число появлений этого цвета в других комбинациях.

Поэтому вы можете немедленно остановить испытание с цветом x в положении i, если:

либо эта комбинация дает слишком много черных шашек по сравнению с другой комбинацией,

либо она еще не дает черных шашек в тот момент, когда остается ровно столько позиций, сколько нужно для создания тех черных шашек, которые остается получить,

либо она дает слишком много белых шашек,

либо она не производит достаточно белых шашек в тот момент, когда остается ровно столько позиций, сколько нужно.

Все это означает, что вместо того, чтобы предлагать новую комбинацию всю целиком, а затем сравнивать ее с уже полученными, вы действуете, перебирая позицию за позицией, и каждое делаемое в этой позиции предложение немедленно интерпретируется для всех уже изученных комбинаций.

Когда мы оказываемся заблокированными, нужно возвращаться назад. Нетрудно скорректировать число появлений рассматриваемых цветов. Нужно переоценить уже полученные числа белых и черных шашек. Вы действуете при этом путем пересчитывания вклада данного цвета в рассматриваемую позицию.

Игра 27.

Рассмотрим игру НАДЕВАТЬ. В начале на игровом поле ничего нет. Можно играть только на поле, которое следует за первым занятым полем, а такого поля нет. Играем на поле 1.

На следующем ходе было бы глупо снимать только что выставленную шашку. Первое занятое поле — первое. Ставим шашку на поле 2. Первое занятое поле — это снова первое поле. Было бы глупо снимать только что выставленную шашку, и поэтому нужно играть на первом поле, т. е. освобождать его. Теперь первое свободное поле — это поле 2. Было бы глупо возвращать на поле 1 только что снятую шашку. Следовательно, поставим шашку на поле 3. Никакого выбора...

Чтобы освободить игру на одно поле, очищаем поле 1.

Чтобы освободить игру на два поля, мы не можем очистить поле 1, так как тогда мы не могли бы очистить и поле 2. Первое занятое поле — поле 1. Можно очистить поле 2, а затем поле 1.

Для игры с 3 полями, мы очищаем 1, затем ставим 3, ставим снова 1, очищаем 2, а затем 1.

Если n четно, то мы начинаем с удаления шашки 2, в противном случае мы удаляем шашку 1.

Теперь вам не составит ни малейшего труда написать итеративную программу:

```
место := 0;   игра := пусто
ВЫПОЛНЯТЬ
   ЕСЛИ поле (1) = пусто ТО поставить (1);
       место := место +1
     ИНАЧЕ удалить (1);
       место := место −1
   КОНЕЦ_ЕСЛИ
   ЕСЛИ место = n ТО КОНЧЕНО КОНЕЦ_ЕСЛИ
   искать первое занятое поле, номер которого дает
   число i;
   ЕСЛИ поле (i + 1) = пусто ТО поставить (i + 1);
       место := место +1
     ИНАЧЕ удалить (i + 1);
       место := место −1
   КОНЕЦ_ЕСЛИ
   ЕСЛИ место = n ТО КОНЧЕНО КОНЕЦ_ЕСЛИ
ВЕРНУТЬСЯ
```

Для игры СНИМАТЬ вы действуете аналогично.

В том, что касается последовательностей чисел, порожденных игрой СНИМАТЬ, начнем с рассмотрения конкретного примера. Вот игра СНИМАТЬ для $n = 4$.

0001 1
0011 3
0010 2
0110 6
0111 7
0101 5
0100 4
1100 12
1101 13
1111 15

Использованы все числа, меньшие 8, а из больших или равных 8 участвуют только 12, 13 и 15. Для обобщения действуйте по индукции.

Игра 29.

Вот решение для 8 букв и 10 полей.

..абабабаб
баабаба..б
бааб..ааб б
б..бааaабб
бббаааа..

Присутствие куска X не меняет последовательности изменений.

..абабXабаб
баабабXа..б
бааб..Xаабб
б..бааXаабб
бббааXаа..

Последний перенос пары букв *аа* слева от X в свободные пары справа дает

бббб..Xаааа

Теперь вы можете заняться X (если для этой комбинации вам решение уже известно) и получить

ббббY..аааа

Таким образом, остается переместить два *а* с крайних полей справа на свободные поля, и все закончено. Следовательно, если вы умеете исследовать комбинацию X с p парами букв *а*, *б*, то вы умеете исследовать и комбинацию с $p + 4$ парами.

Я уже предложил вам решение для четырех пар. Таким образом вы получаете решение для 8, 12,...

Главные решения — это решения для 4, 5, 6, 7 пар. Вот одно из решений для строчки из 5 пар

..абабабабаб

Искомое расположение имеет вид

бббббааааа..

Можно задаться целью удалить все буквы *а* (особенную трудность при перемещениях вызывает то, что их число нечетно) из первой половины (первых 5 позиций, в которых букв *а* в исходном положении не столько же, сколько букв *б*).

..абабабабаб
баабабаба..б
бааб..абаабб
бааббаа..абб
б..ббааааабб
бббббааааа..

Предлагаю вам разыграть 6 и 7 пар.

Совершенно бесполезно подключать к этому делу компьютер. А где же программирование, спросите вы? Я отвечу, что это упражнение вводит вас в рекурсивные или индуктивные рассуждения. Это оздоровляет наши способы рассуждать...

Игра 30.

Единственная настоящая задача, если вы работаете итеративным способом — организовать испытания так, чтобы иметь возможность совершенно систематически проводить их и обновлять игру, сохраняя список ходов, чтобы иметь возможность вернуться назад.

Игра имеет ту же конфигурацию, что и для лис и кур. Поле обозначается своим положением в цепочке. Перемещение в данном направлении реализуется добавлением некоторой константы к данному положению. Таблица из четырех элементов дает эти константы для всех четырех направлений. Свободное поле представляется точкой, занятое поле — крестиком ×.

Вы ищете первый крестик в цепочке и вы начинаете с первого возможного перемещения $i = 1$. Если есть возможность взятия в этом направлении, то вы регистрируете данные ×, i в цепочке или таблице (во втором случае вы симулируете кучу). Вы выполняете взятие и на-

чинаете сначала. Если же возможности взятия в данном направлении нет, то вы переходите к следующему i. Если вы достигаете четырех, то с этим крестиком все кончено, и вы переходите к следующему. Если их больше нет, то вы возвращаетесь к последней зарегистрированной в куче паре данных $\times$, i, отменяете соответствующее взятие (изменение состояния игры) и продолжаете переходом к следующему i.

Вы уже проделали более трудную работу для самого длинного взятия в игре с курами и лисами.

Игра 31.

Число ходов $f(p)$ для переноса p дисков получается переносом сначала $p - 1$ дисков со стержня d на стержень $3 - a - d$ за $f(p - 1)$ ход, затем из перемещения диска p, что требует в точности одного хода, а затем возвращения $p - 1$ дисков из запаса на стержень прибытия за $f(p - 1)$ ход, откуда получаем:

$$f(p) = 2 * f(p - 1) + 1,$$
$$g(p) = f(p) + 1 = 2 * f(p - 1) + 2 = 2 * (f(p - 1) + 1) = 2 * g(p - 1).$$

По индукции

$$g(p) = 2^p g(0).$$

Так как $f(0) = 0$, $g(0) = f(0) + 1 = 1$, $g(p) = 2^p$, то, наконец

$$f(p) = 2^p - 1.$$

Для игры с 50 дисками нужно $2^{50} - 1$ ходов. Но 2^{10} равно 1024, или порядка 10^3. Следовательно, 2^{50} порядка 10^{15}.

В часе 3600 секунд, в сутках $3600 \times 24 = 86400$ секунд, за год получаем 86400×365 — или порядка 3×10^7 секунд, откуда, наконец, 3×10^9 секунд за столетие. Поэтому нужно порядка $10^{15}/3 \times 10^9$, или порядка 3×10^5 веков для игры с 50 дисками, которая, таким образом, требует около 300000 веков...

Вот другой способ доказывать свойство четности. Пусть диски обозначены их порядковыми номерами, начиная с первого — самого маленького, и нужно показать, что два диска с номерами одной четности никогда не попадают непосредственно один на другой.

Опыт показывает, что для первых значений n реализация игры $H(n, d, a)$ дает следующее:

— диски, попадающие в основание стержней d и a, имеют ту же четность, что и n,

— диски, попадающие в основание запасного стержня, имеют другую четность.

Предположим, что это свойство справедливо для $n - 1$. Для реализации $H(n, d, a)$ нужно выполнить сначала $H(n-1, d, 3-a-d)$. В течение этой операции диск n остается в основании начального стержня d и, следовательно, в основании диска d находится диск n и потому диск той же четности, что и n. Диски, которые при этом оказываются в основании стержня прибытия для процедуры $H(n-1, d, 3-a-d)$, имеют (по предположению индукции) ту же четность, что и $n-1$. Но этот стержень прибытия является для игры $H(n, d, a)$ запасным стержнем, и, следовательно, в основании запасного стержня оказываются диски, имеющие ту же четность, что и $n-1$. Наконец, запасной стержень для игры $H(n-1, d, 3-a-d)$ есть a, в основание которого попадают диски с четностью $n-2$, следовательно, с четностью n.

Перемещение диска n со стержня d на стержень a помещает n в основание стержня a, так что при этом свойство четности для a подтверждается. Проверьте, что для стержней d и $3-a-d$ оно также подтверждается.

Для этого разложите $H(n, d, a)$ на 5 операций:

$H(n-2, d, a)$ n и $n-1$ на стержне d
$P(n-1, d, 3-a-d)$ n на d, $n-1$ на $3-a-d$
$H(n-2, a, 3-a-d)$
$P(n, d, a)$ n на a, $n-1$ на $3-a-d$
$H(n-2, 3-a-d, d)$
$P(n-1, 3-a-d, a)$ n на a, $n-1$ на a
$H(n-2, d, a)$.

Предположим, что искомое свойство четности выполняется для $n-1$. Тогда остается заниматься только теми дисками, которые ложатся на диск n.

В первой операции диск $n-1$ находится на диске n, они разной четности, и, таким образом, здесь свойство четности выполняется. Во время игры $H(n-2, a, 3-a-d)$ диск n находится на стержне, который для этой игры является запасным. Диски, которые в этой игре ложатся в основание этого стержня — и потому ложатся на диск n — имеют четность, противоположную четности числа $n-2$, следовательно, четность, противоположную

четности n, что и проверяет на этом этапе наше условие четности. Вы легко завершите это рассуждение.

Разобранный пример хорошо иллюстрирует тесную связь между рекурсивностью и рекуррентностью, которые представляют собою не что иное, как две немного отличающиеся реализации одного и того же рассуждения.

Игра 33.

Предположите, что в $H(n-1, d, a)$ диск 1 перемещается всегда в одном и том же направлении. Для $H(n, d, a)$ вы должны выполнить

$H(n-1, d, 3-a-d)$
$H(n-1, 3-a-d, a)$.

Вместо того, чтобы непосредственно переходить от d к a, вы осуществляете этот переход с помощью стержня $3-a-d$, иначе говоря, вы делаете два перемещения в обратном направлении. Диск 1 продолжает перемещаться всегда в одном и том же направлении, но это направление меняется при переходе от $n-1$ к n. Для $n=1$ этот диск перемещается в направлении от d к a. Это всегда будет так для всех нечетных n, в то время как для четных n он будет перемещаться в направлении от a к d.

Простое итеративное решение имеет следующий вид: исходя из четности n определите направление перемещения диска 1. Начните с 2^n-1 число ходов, которые осталось сделать:

```
s := ЕСЛИ четно (n) ТО 2 ИНАЧЕ 1 КОНЕЦ_ЕСЛИ
d := 0;  k := 2^n − 1
ВЫПОЛНЯТЬ
    a := d + s ; ЕСЛИ a > 2 ТО a := a − 3
    КОНЕЦ_ЕСЛИ
    переместить диск 1 с d на a;
    d := a; k := k − 1
    ЕСЛИ k = 0 ТО КОНЧЕНО КОНЕЦ_ЕСЛИ
    переместить единственный диск, который можно
    переместить, кроме диска 1
    k := k − 1
ВЕРНУТЬСЯ
```

Все диски имеют общее свойство: нечетные диски перемещаются в том же направлении, что и диск 1, а четные диски — в другом направлении.

В вышеприведенной программе стратегия совершенна с точки зрения исполнения вручную, потому что в каж-

дый данный момент сразу видно, какой диск нужно переместить, если это не самый маленький диск (меньший из двух остальных дисков перемещается на больший). В нашей программе вам нужно вычислить это движение. Один из наиболее простых способов состоит в том, чтобы представить игру с помощью вектора, дающего для диска i номер стержня, на котором он находится. Диск, подлежащий перемещению — это наименьший диск, который находится не на том же стержне, что и диск 1, следовательно, номер стержня которого отличается от d. Этот самый диск перемещается со стержня, на котором он находится — с номером x — на стержень $3 - x - d$.

Обозначим первое перемещение через 1. Поскольку диск 1 перемещается один раз в каждой паре ходов (точнее, перемещается через ход), то он перемещается в каждый нечетный ход. По индукции покажите, что диск p перемещается в ходы с номерами, которые делятся на 2^{p-1}, но не делятся на 2^p (т. е. являются нечетными кратными числа 2^{p-1}).

Номер k любого хода может быть единственным способом представлен в виде

$$k = (2r + 1)2^{p-1}.$$

Перемещаемый на этом ходе диск есть диск с номером p, и это — его $(r + 1)$-е перемещение. Так как он начинает движение со стержня 0 и перемещается в направлении s_p (1, если p нечетно, и 2 в противном случае), то на этом ходе диск перемещается с rs_p-го на $(r + 1)s_p$-й стержень, где эти числа берутся по модулю 3.

Игра 34.

Попытаемся охарактеризовать значение p, дающее игре оптимум для данного n. Нам известно, что $f_3(n - p) = 2^{n-p} - 1$.

Должно выполняться

$$2f_4(p - 1) + 2^{n-p+1} - 1 \geqslant 2f_4(p) + 2^{n-p} - 1,$$
$$2f_4(p + 1) + 2^{n-p-1} - 1 \geqslant 2f_4(p) + 2^{n-p} - 1.$$

Удобно пользоваться первыми разностями для функции f_4:

$$d(p) = f_4(p + 1) - f_4(p).$$

Два приведенных выше соотношения могут быть переписаны следующим образом:

$$d(p - 1) < 2^{n-p-1}, \quad d(p) \geqslant 2^{n-p-2}.$$

Интересно рассматривать даже не $d(p)$, а скорее $2^p d(p) = g(p)$:

$$g(p-1) \sim 2^{n-2} \leqslant g(p).$$

Можно еще упростить запись, беря не $g(p)$, а величину

$$h(p) = \log_2 (g(p)) = p + \log_2 (d(p)).$$

Тогда получаем

$$h(p-1) < n-2 \leqslant h(p).$$

При данном n величина p — наименьшее целое, для которого h больше или равно $n-2$.

Приведем здесь первые из полученных таким образом значений:

n	q	f_4	p	d	h
0	0	0		1	0
1	1	1		2	2
2		3		2	3
3	2	5	1	4	5
4		9	1	4	6
5		13	1	4	7
6	3	17	3	8	9
7		25	3	8	10
8		33	4	8	11
9		41	5	8	12
10	4	49	6	16	14
11		65	6	16	15

Мы добавили в таблицу переменное q, связанное с «треугольными» числами. Для $n = q(q+1)/2$ действительно убеждаемся, что

$$h(p) = h(p-1) + 2$$

в то время как для других n

$$h(p) = h(p-1) + 1.$$

Исходя из n, можно вычислить q:

$$q = \text{целая_часть}\ ((\sqrt{8n+1} - 1)/2).$$

Имеем

$$h(n) = n + \text{целая_часть}\ ((\sqrt{8n+1} - 1)/2).$$

Покажите это по индукции. Исходя отсюда, вычисляется все. Таким образом, если n дано, то p — наимень-

шее целое, большее или равное

$$(2n - 1 - \sqrt{8n - 7})/2.$$

Игра 35.

Возьмем, например, игру с 50 дисками. Она реализуется переносом сначала 40 дисков на запасной стержень, а затем 10 последних дисков со стержня 0 на стержень 1, с использованием при этом только трех свободных стержней. Наконец, остается перенести начальные 40 самых маленьких дисков с запасного стержня на первый стержень, используя все 4 стержня.

Чтобы переместить 40 дисков с 4 стержнями, сводим задачу к перемещению 31 диска с 4 стержнями, а затем 9 с 3 стержнями...

Таким образом, дело сводится к разбиению 50 дисков на 8 сегментов:

сегмент 1:	1 диск	номер диска	1	
2	3 диска	диски от	2 до	4
3	6		5	10
4	6		11	16
5	7		17	23
6	8		24	31
7	9		32	40
8	10		41	50

Каждый сегмент перемещается с использованием 3 стержней, в чем мы следуем итеративной стратегии, которая уже описана выше. Единственный вопрос — это правильный выбор запасных стержней.

Договоримся работать с тремя стержнями 0, 1, 2, так что стержень 3 остается пустым и служит запасным стержнем при любом перемещении какого-либо сегмента. Более точно, перемещение сегмента p со стержня d на стержень a осуществляется с помощью изученной выше процедуры H, в которой запасным стержнем является стержень 3.

Сегмент 1 перемещается в каждый из двух ходов подряд (под ходом я понимаю последовательность операций, реализующих процедуру H), всегда в одном и том же направлении.

Мы сохраняем предыдущую итеративную стратегию, но понимаем ее как стратегию для сегментов. На компьютере это может пройти очень быстро. Вполне вероятно, что робот может осуществить одно перемещение за

несколько секунд. Тогда на всю игру потребуется не более чем несколько часов...

Игра 36.

Соотношение $SG(p, q) = 0$ означает, что вы не можете достичь ситуации с числом Спрага—Грюнди, равным нулю, удаляя не более $2q$ спичек из кучки с p спичками. Если вы исходите из $SG(p, q' < q)$, то вы не можете удалить столько же спичек и, следовательно, нет опасности, что вы получите число SG, равное нулю.

Предположим, что $SG(p_i, 1) = 0$.

Исходя из $p_i + 1$, я могу удалить 1 спичку и получить пару p_i, 1. Следовательно, $SG(p_i + 1, q) \neq 0$.

Исходя из $p_i + 2$, я для любого q всегда могу удалить две спички, но тогда я получаю $SG(p_i, 2) \neq 0$, и, следовательно,

$$SG(p_i + 2, 1) = 0.$$

Если в p_i имеем $q_i > 1$, то тогда мы этого не получим и $SG(p_i + 2, 1) \neq 0$. Но в $p_i + 3$, удаляя единственную спичку, получаем пару $p_i + 2$, 1 с $SG \neq 0$, или же, удаляя две спички, получаем пару $p_i + 1$, 2 с ненулевым числом SG. Следовательно, $SG(p_i + 3, 1) = 0$.

Все оставшееся уже очень хорошо подготовлено. Рассмотрите точку p, для которой диагональ пересекает ось $p = 0$, не пересекая положений с нулевым SG. Эта прямая задается уравнением $x + y = p$. Она пересекает ось $x = 0$ в точке $y = p$. Нельзя взять в точности p спичек,— можно не больше $p - 1$. Следовательно, в этой точке

$$q = \text{целая_часть}\ ((p - 1)/2).$$

Рассмотрим теперь точку, абсцисса которой есть число Фибоначчи: $p = \text{fib}\ (s)$.

Нужно показать, что прямая $x + y = \text{fib}\ (s)$ не пересекает точек с ненулевыми SG, кроме $x = 0$. Рассмотрим сначала точку

$$x = \text{fib}\ (s - 1).$$

В этой точке

$$y = \text{fib}\ (s) - \text{fib}\ (s - 1) = \text{fib}\ (s - 2).$$

При $p = \text{fib}\ (s - 1)$

$$q = \text{целая_часть}\ ((\text{fib}\ (s - 1) - 1)/2).$$

Нужно показать, что для каждого s

$$\text{целая_часть}\ ((\text{fib}\ (s - 1) - 1)/2) < \text{fib}\ (s - 2),$$

или, что равносильно,

$$\mathrm{fib}\,(s-1) < 2 * \mathrm{fib}\,(s-2) + 1.$$

Но

$$\mathrm{fib}\,(s-1) = \mathrm{fib}\,(s-2) + \mathrm{fib}\,(s-3)$$

и

$$\mathrm{fib}\,(s-3) < \mathrm{fib}\,(s-2).$$

Следовательно, рассматриваемая диагональ не пересекает точек с нулевым SG в fib $(s-1)$. Она не может пересекать их и между $s-1$ и s, поскольку эта часть воспроизводит то, что происходит в интервале от 1 до fib $(s-2)$, а диагональ, выходящая из fib $(s-2)$, не пересекает точек с нулевым SG до оси q.

Вы без труда завершите это доказательство.

6. КОМБИНАТОРНЫЕ ЗАДАЧИ

Головоломка 28.

Действительно ли вам что-то еще нужно сообщать? Тогда я немного уточню способ поддержания части от 1 до p в порядке неубывания. Исходим из упорядоченного по неубыванию вектора $a_1, a_2, \ldots, a_p$. Вы последовательно заменяете элемент a_p элементами a_i, где i направлен по убыванию. Вы последовательно получите

$a_1, a_2, \ldots, a_{p-1}, a_p,$
$a_1, a_2, \ldots, a_p, a_{p-1},$
$a_1, a_2, \ldots, a_{p-2}, a_{p-1}, a_p, a_{p-2}.$

По индукции, предположим, что в некоторый момент вы получили

$a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_p, a_i$

после перемены мест элементов с номерами i, p.

На следующем ходе вы поменяете местами a_{i-1} и последний член, который равен a_i. Эта форма остается неизменной, и первая часть, от 1 до $p-1$, остается отсортированной в неубывающем порядке. В конце вы получите

$a_2, a_3, \ldots, a_p, a_1.$

Чтобы восстановить исходный порядок, вы располагаете последний элемент на запасном поле, поднимаете все остальные элементы на одну ступень, а затем размещаете содержимое запасного поля на первом месте.

Это вы делаете только в случае необходимости. Незачем восстанавливать порядок, когда все закончено.

Процедура работает достаточно быстро для того, чтобы в случае неудачи иметь возможность испытать наличие решения для $n-1$, а затем для $n+1$. Таким образом, по прошествии 45 с для каждого кандидата мы получаем в качестве результата

— решение, если оно существует,

— приближение с точностью до единицы, если это возможно.

Эта программа терпит неудачу крайне редко.

В выпуске от 8 марта 1984 года следующий розыгрыш не был найден ни кандидатами, ни Бертраном, ни кем-либо из присутствующих:

50 10 10 5 4 2 $n = 767$.

На моем микрокомпьютере нужно 18 с, чтобы обнаружить, что эта задача не имеет решения, а затем еще 5 с, чтобы получить

$50-10=40$, $40*5=200$, $10-2=8$,
$200-8=192$, $192*4=768$.

Для задачи

9 7 6 4 3 1 $n = 795$

через 6 с получается

$4*9=36$, $36+1=37$, $37*7=259$,
$259+6=265$, $265*3=795$.

Наконец,

100 50 8 5 4 2 $n = 631$.

За менее чем 2 с получаем

$9250-50=9200$,
$9200/25=368$.

7. ОБО ВСЕМ ПОНЕМНОГУ

Головоломка 31.

Программисты обманываются, поскольку они не берут на себя труд прояснить различные ситуации.

В строке 200 мы знаем, что цепочка a пройдена полностью, и исследованы все символы, не являющиеся пробелами. Если в цепочке b содержится еще какой-нибудь символ, не являющийся пробелом, то равенства цепочек нет. Все в порядке.

В строке 300 цепочка a пройдена вплоть до некоторого символа, не являющегося пробелом, и этот символ еще не исследован. Цепочка b пройдена полностью, и в ней не содержится более ни одного символа, не являющегося пробелом. Следовательно, эти две цепочки различны. Можно было бы сказать, что дальнейшее движение по цепочке a бесполезно, но не приводит к ошибке. Но это неверно. Вы остановились на еще не исследованном символе, который не является пробелом. Если вы перейдете к следующему символу, не являющемуся пробелом, то данный символ вы потеряете. Если, как бывает в большинстве случаев, цепочка a совпадает с цепочкой b с точностью до пробелов за исключением единственного дополнительного символа в конце цепочки, то именно по этой-то причине и должен быть остановлен пробег цепочки a. Перемещаясь и не обнаруживая больше символов, не являющихся пробелами, мы получаем сообщение, что цепочки совпадают, а это неверно. Ясио, что программисты не принимали во внимание и не изучали именно этот случай. И никаких оснований поступать так нет. В этом и состоит преимущество логических рассуждений о тексте программы по сравнению с проверкой ее правильности с помощью тестирования.

Ваша программа должна сохранять симметричную роль обеих цепочек. Не начинайте проверять результат пробега цепочки a, не пробежав цепочки b, и изучайте обе цепочки разом.

Возьмем общую ситуацию:

a пройдена вплоть до i включительно;
b пройдена вплоть до j включительно;
обе части совпадают с точностью до пробелов.
ВЫПОЛНЯТЬ
    продвинуть i на следующий символ в a, не являющийся пробелом;
    продвинуть j на следующий символ в b, не являющийся пробелом;
    ЕСЛИ таких нет в a И таких нет в b ТО $r :=$ ИСТИНА;
      КОНЧЕНО КОНЕЦ_ЕСЛИ;
    ЕСЛИ таких нет в a ИЛИ таких нет в b ТО $r :=$ ЛОЖЬ;
      КОНЧЕНО КОНЕЦ_ЕСЛИ;
    ЕСЛИ $a[i] \neq b[j]$ ТО $r :=$ ЛОЖЬ;
    КОНЧЕНО КОНЕЦ_ЕСЛИ;
ВЕРНУТЬСЯ

Эта программа совершенно симметрична относительно a и b...

Головоломка 33.

Нужно работать по модулю n. Удобнее всего пронумеровать элементы вектора от 0 до $n-1$. Все элементы спускаются вниз на m по модулю n. Элемент, который переходит в 0, имеет номер m; элемент, который переходит в m, имеет номер $2m$ по модулю n; элемент, который переходит в $2m$, имеет номер $3m$ по модулю n... Таким образом, мы получаем цепочку чисел, кратных m по модулю n. Весь вопрос в том, чтобы узнать, порождает ли последовательность чисел, кратных m по модулю n, последовательность всех целых от 0 до $n-1$.

Это так, если m и n взаимно просты. В противном случае пусть c — наибольший общий делитель m и n:

$$m = m'c, \quad n = n'c,$$
$$n' * m = n' * m' * c = m' * n = 0 \text{ по модулю } n.$$

Эта цепочка возвращается в 0 за $n' = n/c$ операций. При этом пробегается не весь вектор, а только его элементы, сравнимые с 0 по модулю c.

Беря в качестве исходных элементов различных циклов последовательно целые числа от 0 до $c-1$, вы разместите все элементы вектора, причем каждый из них будет перемещаться в точности один раз...

Головоломка 34.

Рассмотрите более общую задачу, что заставит вас открыть одно из этих знаменитых «преобразований программы», столь полезных, когда желательно улучшить уже существующие программы. Обозначим через t и u два условия, а через a и b — две последовательности инструкций. Вот простой цикл:

```
ПОКА t ВЫПОЛНЯТЬ
   ЕСЛИ u ТО a ИНАЧЕ b
   КОНЕЦ_ЕСЛИ
ВЕРНУТЬСЯ
```

$50 - 4 = 46$, $46 * 2 = 92$, $92 * 8 = 736$,
$100 + 5 = 105$, $736 - 105 = 631$.

Я уже предлагал вам следующий пример:

100 75 50 25 10 10.

Для $n = 370$ особой трудности нет, потому что это — кратное десяти.

Компьютер сообщает мне

75/25 = 3,
50 — 3 = 47,
47 * 10 = 470,
470 — 100 = 370.

Это уже интересно, потому что это — совершенно не то решение, которое я собирался искать.

Чтобы найти 369, нужно образовать число, не кратное 5,— чего нельзя сделать с помощью какой-либо иэ трех операций +, —, *, сохраняющих кратность пяти. Следовательно, нужно использовать деление. Вот решение:

50/10 = 5,
5 * 75 = 375,
375 — 10=365,
100/25 = 4,
365 + 4 = 369.

Обе представленные здесь программы не позволяют получить это решение. Действительно, оно записывается в виде

(50/10) * 75 + 100/25 — 10.

А число 368? Вы нашли для него решение? Я не сумел. Но Жак Бейгбеде сообщил мне, что он получил его делением на 25...

10 * 100 = 1000,
1000 — 75 = 925,
925 * 10 = 9250.

Последовательность операций следующая:

— проверяется условие t,

— если оно истинно, то проверяется u,

— если u истинно, то выполняется a, и все возобновляется.

Допустим, что условия t и u таковы, что я имею возможность проверить u, даже если проверка условия t дает значение ЛОЖЬ*). Тогда, пока условия t и u истинны, в цикле выполняется a.

Вот другая последовательность, которая может встретиться:

— проверяется условие t,

— если оно истинно, то проверяется u,

---

*) Вот пара условий, которая не обладает этим свойством: t: $x \neq 0$; u: $\sin(1/x) > 0$.— *Примеч. ред.*

— если u ложно, то выполняется b, и все возобновляется.

Таким образом, мы приходим к форме, для которой можно доказать, что она всегда эквивалентна исходной (с точностью до ограничения, что должна существовать возможность вычисления u даже в случае, когда t ложно).

```
ПОКА t ВЫПОЛНЯТЬ
   ПОКА t И u ВЫПОЛНЯТЬ a ВЕРНУТЬСЯ
   ПОКА t И НЕ u ВЫПОЛНЯТЬ b ВЕРНУТЬСЯ
ВЕРНУТЬСЯ
```

Мы перепишем программу для определения равнин, чтобы придать ей форму ПОКА, заключенного в скобки ЕСЛИ:

```
i := 1;  p := 0;
ПОКА i ⩽ n ВЫПОЛНЯТЬ
   ЕСЛИ a [i] = a [i − p]
     ТО x := a [i]; p := p + 1; i := i + 1
     ИНАЧЕ i := i + 1
   КОНЕЦ_ЕСЛИ
ВЕРНУТЬСЯ
```

Мы обнаруживаем, что в нашем случае мы не можем объединить два условия с помощью операции И: если i не удовлеворяет условию, что i не больше n, то нельзя поставить вопрос относительно $a[i]$. Обрисуем трудность подходящим образом:

— нужно либо добавить в таблицу a поле, которое содержит какую-нибудь несущественную для нас величину (мы к этой величине не обращаемся);

— либо нужно допустить, что операция И не коммутативна. Для вычисления t и u мы вычисляем t, и если результат есть ЛОЖЬ, то все кончено и притом с результатом ЛОЖЬ. В противном случае результат есть значение условия u.

Тогда можно использовать наше преобразование:

```
i := 1;  p := 0;
ПОКА i ⩽ n ВЫПОЛНЯТЬ
   ПОКА i ⩽ n И a [ i ] = a [ i − p] ВЫПОЛНЯТЬ
     x := a [i]; p := p + 1; i := i + 1
   ВЕРНУТЬСЯ
   ПОКА i ⩽ n И  a [i] ≠ a [ i − p ] ВЫПОЛНЯТЬ
     i := i + 1
   ВЕРНУТЬСЯ
ВЕРНУТЬСЯ
```

Первый цикл движется по таблице a, пока обнаруживается, что элементы равны между собой. Более точно, p и i изменяются одинаково, так что разность $i - p$ остается постоянной. Все элементы $a[i]$ сравниваются с одним и тем же элементом, и величина x остается постоянной, равной этому элементу, на протяжении всего цикла.

Второй цикл изменяет i до тех пор, пока не обнаружится пара элементов, отстоящих на $p + 1$.

Уточним ситуацию выхода из первого внутреннего цикла. Мы собираемся найти конец равнины, которая лучше всех предыдущих, мы фиксируем ее длину p и ее значение x, а i обозначает первый элемент после этой равнины. Мы можем надеяться найти пару j, $j - p$ с

$$a[j] = a[j - p]$$

только пока $j - p$ остается на равнине, которую мы собираемся пройти. Наименьшее соответствующее i значение j удовлетворяет условию $j - p = i$, или $j = i + p$.

Следовательно, можно увеличивать i от p в обоих циклах, не меняя действия программы, что ускоряет ее работу.

Чтобы ускорить и первый внутренний цикл, мы присвоим переменной x ее значение перед циклом и сохраним ее начальное значение в j. Так как $i - p$ остается постоянным, то можно вычислить значение p также и после выхода из цикла. Начальные значения суть $i = j$ и $p = p_0$, а конечные значения i и p удовлетворяют соотношениям $i - p = j - p_0$, откуда $p = i + p_0 - j$:

```
i :=1; p :=0
ПОКА i ⩽ n ВЫПОЛНЯТЬ
    x := a[i]; j := i
    ПОКА i ⩽ n и a[i] = x ВЫПОЛНЯТЬ
        i := i + 1
    ВЕРНУТЬСЯ
    p := i + p − j; i := i + p
    ПОКА i ⩽ n И a[i] ≠ a[i − p] ВЫПОЛНЯТЬ
        i := i + 1
    ВЕРНУТЬСЯ
ВЕРНУТЬСЯ
```

Вы можете получить эту программу непосредственно, минуя механизм преобразования программ. Но этот способ кажется мне требующим больших умственных усилий.

Может быть, это связано с ходом мыслей, который я приобрел, преподавая *).

Головоломка 35.

Хорошенько учтите то, что вы знаете: обозначим через u таблицу, которая дает последние элементы наилучших возрастающих последовательностей для (всех возможных) длин от 1 до m.

Покажем сначала, что $u_i < u_{i+1}$. Предположим, что это не так: пусть существует такая последовательность длины $i+1$, у которой последний элемент не больше u_i.

---

*) Прочтя весь этот ужас, я решил провести решение, основанное на методике из курса программирования механико-математического факультета МГУ.

Каждой последовательности чисел $\{a_1, a_2, \ldots, a_i\}$ $(i \geqslant 1)$ сопоставим число lmax, равное максимальной длине равнинного участка этой последовательности. Очевидно, что lmax $(\{a_1\}) = 1$. Пусть мы знаем lmax $(\{a_1, a_2, \ldots, a_i\})$. Как вычислить величину lmax $(\{a_1, \ldots, a_i, a_{i+1}\})$? Добавление элемента a_{i+1} к последовательности $\{a_1, a_2, \ldots, a_i\}$ не затрагивает равнинных участков этой последовательности, кроме, быть может, последнего. Если $a_{i+1} = a_i$, то длина этого последнего участка — назовем ее llast $(\{a_1, \ldots, a_i\})$ — увеличивается на единицу. Если величина llast $(\{a_1, \ldots, a_i, a_{i+1}\})$ окажется при этом больше величины lmax $\{a_1, a_2, \ldots, a_i\}$, то это значит, что последний равнинный участок в последовательности $\{a_1, a_2, \ldots, a_i, a_{i+1}\}$ по крайней мере на 1 длиннее всех предыдущих, и, значит, lmax $(\{a_1, a_2, \ldots$ $\ldots, a_i, a_{i+1}\})$ = llast $(\{a_1, a_2, \ldots, a_i, a_{i+1}\})$.

Введем четыре величины:

i — число рассмотренных членов последовательности,

lmax — максимальная длина равнинного участка для рассмотренных элементов,

llast — длина последнего равнинного участка для рассмотренных элементов,

xlast — последний рассмотренный элемент последовательности (он равен $a[i]$).

Теперь приведем без пояснений программу, которая вычисляет lmax $(\{a_1, \ldots, a_n\})$ по индукции.

```
i := 1;  lmax := 1;  llast := 1;  xlast := a [1]
нц пока i < n
  x := a [i + 1]
  если x = xlast то llast := llast + 1
  иначе llast := 1 кесли
  если llast > lmax то lmax := llast кесли
  xlast := x
  i := i + 1
кц
вывод lmax
```

Подробнее об этой индуктивной методике можно прочитать в книге: А. Г. Кушниренко, Г. В. Лебедев. Программирование для математиков.— М.: Наука, 1988.— *Примеч. ред.*

Так как эта последовательность возрастает, то ее предпоследний элемент меньше u_{i+1} и потому меньше u_i. Тогда, удаляя последний элемент этой последовательности, мы получили бы последовательность длины i с последним членом, меньшим u_i, что противоречило бы предположению, что u_i — последний элемент последовательности длины i с наименьшим возможным последним элементом.

Рассмотрим теперь следующий элемент x нашего вектора. Разместим его в упорядоченной таблице u. Может случиться, что $x > u_m$. Тогда элемент x можно присоединить к концу последовательности длины m; тем самым получилась бы (впервые) возрастающая последовательность длины $m + 1$, которая вследствие своей единственности была бы оптимальна.

Если x меньше u_1, то им следует заменить u_1 для построения новой наилучшей последовательности с длиной 1. Если же, наконец, оказывается, что $u_i < x < u_{i+1}$, то x можно присоединить к концу последовательности с длиной $i + 1$, чтобы получить последовательность с длиной $i + 1$, которая лучше уже известной, и поэтому u_{i+1} следует заменить на x. Так как u упорядочена, то вы можете разместить в ней x с помощью дихотомического поиска.

Эта операция требует порядка $\log_2 m$ действий для m, не превосходящих n. Так как вам требуется n обращений к таблице, то вы получаете верхнюю границу числа действий порядка $n \log_2 n$, что чрезмерно завышено.

Головоломка 36.

Предположим, что вы уже прошли первую цепочку вплоть до индекса $i - 1$ и получили наилучшие слова длины p, меняющейся от 1 до m. Вы рассматриваете символ в положении i и ищете его в другой цепочке. Его первое положение j_1 может быть поставлено в конце некоторого слова — скажем, слова длины p_1 — и даст слово длины $p_1 + 1$, которое окажется лучшим, чем предыдущее: действительно, если j_1 можно поставить после слова длины p_1, то это значит, что его значение больше положения последнего символа в наилучшем слове длины p_1, но меньше положения последнего символа в слове длины $p_1 + 1$. Рассмотрим теперь второе появление того же символа во второй цепочке: $j_2 > j_1$. Его нельзя поставить в конце слова длины $p_1 + 1$, хотя j_2 и больше j_1, потому что это — другое появление того же символа, и их не нужно смешивать. Поэтому достаточно ограничиться по поводу этого появления символа обращением к таблице в ее части от $p_1 + 2$ до m.

Головоломка 37.

Рассмотрим прямоугольник пробелов, вертикальная граница которого расположена в столбце j и располагающийся вправо от этого столбца в строках от i_1 до i_2. Его основание равно $\inf(l[i_1 : i_2])$, а его площадь есть произведение этого основания на его высоту $i_2 - i_1 + 1$.

Для столбца j нужно найти максимум этой величины, когда i_1 меняется от 1 до $n - 1$ (n — число строк), а i_2 — от $i_1 + 1$ до n.

Когда вы переходите к следующему столбцу, то каждое l уменьшается на 1. В строке, в которой стояла единица, оно становится нулем. Там, где l было равно 0, его нужно вычислить заново. Вы можете попробовать схитрить при вычислении величины $\inf(l)$.

В центральном цикле любое введение нового члена может только уменьшить значение минимума.

Головоломка 39.

Рассмотрим значения S для строк i и $i' > i$. Очевидно

$$S(i, j) = S(i, i' - 1) + S(i', j).$$

Если $S(i, i' - 1)$ положительно, то $S(i, j) > S(i', j)$ и строка i остается строкой, которая может содержать максимум.

Но если $S(i, i' - 1) < 0$, то $S(i, j) < S(i', j)$.

Максимум нужно тогда искать либо среди $S(i, j)$ для $j < i'$, либо среди $S(i', j)$ для $j \geqslant i'$.

Заметим, что $S(i', i') = a[i']$.

Мы собираемся пробежать строку $S(1, \ldots)$ вплоть до первого индекса i_1, для которого S становится отрицательным. Тогда мы начнем пробегать строку $S(i_1 + 1, \ldots)$, и т. д.

Отсюда следует, что в каждый данный момент нужно знать максимальную подпоследовательность в уже пройденной части; эта подпоследовательность задается номером начала r, номером конца q и своей суммой m. С другой стороны, нужно знать наилучшую заключительную подпоследовательность $S(k, i - 1)$, предполагая, что вектор пройден вплоть до поля $i - 1$. Обозначим через s значение суммы этой заключительной подпоследовательности. Пусть k — номер строки, дающий этой сумме максимальное значение, а s — сумма всех членов, начиная с k.

Если сумма s положительна, то она и образует максимум на строке с номером k. При переходе к i число $a[i]$ добавляется к s. Если s отрицательно, то новый элемент с номером i и становится оптимальной строчкой, и нужно взять $s = a[i]$.

В этих двух случаях число s нужно сравнить с оптимумом m. Если s оказывается больше, то m нужно заменить на s. Попытаемся составить программу, исходя из того, что мы сейчас обсудим. Нужно уточнить предположение индукции.

Предположим, что вектор пройден от элемента 1 до элемента с номером $i - 1$ включительно. Мы знаем лучшую подпоследовательность в этой части: она идет от индекса r до индекса q включительно, и ее сумма равна $m : m = S(r, q)$. С другой стороны, мы знаем наилучшую заключительную подпоследовательность, кончающуюся в $i - 1$, т. е. знаем такой индекс k, что сумма $S(k, i - 1)$ максимальна среди заключительных подпоследовательностей. Значение суммы $S(k, i - 1)$ равно s. Может случиться, что эта заключительная подпоследовательность является наилучшей возможной во всей пройденной части, и в этом случае имеем $r = k$, $q = i - 1$, $s = m$. В любом другом случае $s \leqslant m$. Если $i = n - 1$, то весь вектор пройден и получен искомый результат r, q, m.

В противном случае нужно включить элемент $a[i]$. Если s отрицательно, то $a[i]$ и образует (как единственный участник) наилучший заключительный отрезок; берем $k = i$, $s = a[i]$. В противном случае $s \geqslant 0$ и сумма $s + a[i]$ больше s и больше $a[i]$, и это и есть сумма для наилучшего заключительного отрезка, который по-прежнему начинается с номера k. В этих двух случаях отрезок s становится наилучшим отрезком, если он оказывается больше m.

Для начала можно положиться на пробег вектора, начиная с его единственного первого элемента. В этот момент наилучший сегмент и наилучший заключительный сегмент — это одно и то же.

```
d := 1; f := 1; m := a[1]; s := m; i := 2
ПОКА i ⩽ n ВЫПОЛНЯТЬ
   ЕСЛИ s < 0 ТО k := i; s := a[i]
      ИНАЧЕ s := s + a[i]
   КОНЕЦ_ЕСЛИ
   ЕСЛИ s > m ТО d := k; f := i; m := s
   КОНЕЦ_ЕСЛИ
   i := i + 1
ВЕРНУТЬСЯ
```

Эта программа осуществляет пробег вектора a один-единственный раз, что и было предписано в условии. Это очень просто, но это совершенно не очевидно.

СПИСОК ЛИТЕРАТУРЫ

Произведения, цитируемые в тексте

[ARS] Arsac J., Les bases de la programmation, Paris, Dunod, 1983.

[BAI] Baillif J.— C., Les casse — tête logiques de Baillif, Paris, Dunod, 1979.

[BAL] Ball W.— W. Rouse, Mathematical recreations and essays, Macmillan and C°, London, 1963.

[BER] Berloquin P., Le jardin du sphynx, Paris, Dunod, 1981.

[ENG] Engel A., Mathématique élémentaire d'un point de vue algorithmique, Paris, Cedic, 1979.

[GRI] Gries D., The science of programming, Springer Verlag, New York, 1981.

[KNU] Knuth D., The art of computer programming, Addison Wasley, 1969.

[KUE] Kuenzi N.— J., Prielipp B. Cryptarithms and other arithmetical pastimes, School science and mathematics association, University of Wisconsin.

[LED] Ledgard H.— F., Proverbes de programmation, Paris, Dunod, 1978.

[PBB] Berlioux P., Bizard Ph., Algorithmique, Paris, Dunod, 1983.

[POL] Pollard J.— M., A Monte Carlo method for factorization, BIT 15, (1975), p. 331—334.

[SIK] Siklossy L., Let's talk Lisp, Prentice Hall, Englewood Cliffs (N. Y.), 1976.

[SCH] Schwartz B., Mathematical solitaires and games, Baywood Publishing Company, 1978.

Для тех, кому нужно пополнить свое образование в программировании.

Arsac — Mondou O., Bourgeois — Camescasse, Gourtay M.

Premier livre de programmation (écriture de boucles de programmes).

Deuxiéme livre de programmation (procédures, fichiers).

Pour aller plus loin en programmation (récursivité, structures de données),

Cedic — Nathan, Paris, 1982.

Taurisson A., Petitguillaume A.

A vous de jouer, Introduction à la science de l'informatique, Modulo Editeur, Outremont, Québec, Canada.

СОДЕРЖАНИЕ